Automatic Recognition and Interpretation of Finite State Automata Diagrams

by

Olusola Tope Babalola

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science in the Faculty of Science at Stellenbosch University

Department of Mathematical Sciences (Computer Science)
Faculty of Science
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Supervisor: Prof. L. van Zijl

December 2015

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....

Dedication

To my parents, siblings, wife, and daughter.

Abstract

An application capable of reading graphically-encoded information is beneficial to blind or visually impaired students. Such a system needs to recognize and understand visual markings and their arrangement as presented in a diagram image. In that light, this thesis examines the practical possibility of a real world system for the automatic recognition and interpretation of machine-printed Finite State Automata diagrams. The suggested system uses known image processing and pattern recognition methods to extract the visual markings from the diagram image pixels. A second stage, to interpret the meaning of the diagram, is based on modeling the language of Finite State Automata diagrams using Constraint Multiset Grammars. Our results show that a practical application for automatic interpretation of Finite State Automata diagrams is possible.

Acknowledgement

I would first like to acknowledge Prof. L. van Zijl for the wonderful support, encouragement, and supervision all through this project; it was a blessing having her as my supervisor. Nothing can repay your efforts.

I would also like to thank my family who bore the financial and emotional costs of postgraduate study, especially my daughter Anu who spent most of her little years on the phone with her dad. Rev. & Mrs. Williams and the family at Destiny Student Ministry deserves appreciation for providing social and spiritual support, our interactions sustained me in high spirits; space will fail me to mention names, everyone there was nice and I never felt I was in a foreign land. The Landons over at Idas Vallei housed me and were kind to me. Chantal Swartz's help at the Postgraduate Office towards securing financial aid for my final registration contributed in a great way to the successful completion of the programme.

My journey to Stellenbosch University started with an encounter with Sunday Adeniyi, who was then a PhD student in the institution, and ended at the Computer Science department; for the opportunity given to me to study in this great place, I am indeed grateful. Thank you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Long before there was written language, there were depictions, of myriad varieties .... Some of these depictions probably had religious significance, but many were used to communicate, to keep track of events in time, to note ownership and transactions of ownership, to map places, and to record songs and sayings ..." [147].

Diagrammatic notations in printed and electronic media are a major part of academic instructional materials. However, access to the information contained in diagrammatic constructs remains a challenge to blind and visually impaired (BVI) students. For BVI students, access to diagrams is typically provided by reproducing images as tactile copies. This approach has many disadvantages, such as the fact that tactile graphics are larger than the original graphics, often flowing into multiple sheets; BVI students require additional skills to read tactile diagrams; and the cost of and time required to reproduce graphics in tactile form are prohibitive. Consequently, the automatic recognition and analysis of visual content by machines offer another avenue to explore. However, graphics recognition technology is yet to reach the maturity of text recognition or screen reader technology. While optical character recognition (OCR) research successfully led to the development of several free and commercial applications, graphics recognition research still has many unsolved problems.

The automated processing of diagrams of any type is challenging, because diagrams are terse, concise and compact in composition, non-linear in formation, and are more heterogeneous than written communication. If a picture is worth a thousand words, it may mean that over a thousand words of verbal communication is embedded in a single image. How the inherent meanings can be correctly extracted from a printed diagram in the presence of noise, ambiguity, and possible structural imperfections from the original document, is therefore quite challenging. Furthermore, the diagram recognition and understanding field is yet to have a standard model for creating recognition systems [94].

It has been noted [14] that diagram recognition and interpretation is an ambitious undertaking, and may well be unachievable given the nature and diversity of diagrams. Diagram types differ structurally, the rules (syntax) of their composition differ, and the meanings (semantics) of the composition also differ. However, focusing recognition efforts on a specific notation offers a more realistic but still challenging goal. Such approaches led to the development of specialized systems which can understand and play music from OCR of sheet music [125], the automatic recognition of chemical formulae [111], and systems that produce computer aided design (CAD) drawings from paper drawings [46, 150]. Similarly, chart recognition and understanding has been examined [78], and UML diagrams recognition [83] has also been undertaken.

In this work, we investigate a practical automated procedure for the recognition of Computer Science diagrams, with the objective of interpreting the visual representations depicted in them. Our interest is in graph-like structures. Graphs are widely used in Computer Science texts and present a notable challenge for BVI students [24]. We concentrate on Finite State Automata (FSA) diagrams; a dialect of graphs but with additional visual and semantic elements. FSA diagrams embed all their semantic characteristics in simple visual structures made up of lines, circles, and text. Our goal is the automatic extraction of the syntax and semantics captured in these diagrams. This operation is an essential step preliminary to the automatic representation of the diagram to a BVI student.

Our research question, then, is to investigate the practical possibility of a real world automated system for the recognition and interpretation of FSA diagrams for BVI users.

In the next chapter, diagram recognition is explored, followed by a review of some existing attempts at interpreting diagrams, and the image processing and pattern recognition techniques used in diagram recognition. Chapter 3 examines the visual structure of FSA diagrams and describes our diagram structure recognition stage. The use of formal languages in the description of visual forms is examined in Chapter 4, thus establishing the foundation for the diagram interpretation stage.

The next stage of the interpretation system applies domain specific knowledge about FSA diagrams to parse the diagram and extract semantic elements. Chapter 5 describes our approach to syntax specification and parsing of FSA diagrams. In Chapter 6, the experiments performed and the results obtained are reviewed. We conclude in Chapter 7.

Chapter 2

Literature survey on image processing techniques and diagram recognition

"... it is necessary, while formulating the problems of which in our further advance we are to find the solutions, to call into council the views of those of our predecessors who have declared any opinion on this subject, in order that we may profit by whatever is sound in their suggestions and avoid their errors." - Aristotle, as cited in [11].

There are several types of diagrams. In this thesis we consider scientific diagrams, in particular diagrams from the field of Computer Science. Diagrams in Computer Science are quite often concerned with the depiction of interactions and interrelationships between entities. As such, they can mostly be classified as graph diagrams consisting of nodes and edges.

The process of automatic recognition of diagrams in printed documents is referred to as diagram recognition. Diagram recognition is a part of document image analysis research and a major research theme in the graphics recognition sub-area of document processing [86]. Graphics recognition deals with the analysis of non-text constituents of a printed page [86], including lines, symbols, line diagrams, logos, and info-graphics.

The computer analysis of a diagram image to extract semantics conveyed in the diagram is a multiphase and multi-strategy process involving different algorithms, methods, techniques, and approaches ([143] lists a number of 'stable, robust, and off-the-shelf' techniques for some common processes required in graphics recognition). The analysis of diagram information from images draws techniques from various fields including image processing and analysis, pattern recognition, formal language theory, and artificial intelligence.

This chapter examines the analysis of diagrams, starting from a printed machine-

drawn diagram. Offline recognition differs from the recognition of sketched diagrams (usually referred to as online recognition [43]). Throughout the chapter, we point out the applicability of a number of analysis techniques to that of FSA diagram analysis.

2.1 Basic image processing concepts and techniques used in graphics recognition

Since diagram recognition is a complex and multifaceted process, several approaches have been adopted for the different tasks involved in solving the problem. The fundamental approaches used for the manipulation of a raw diagram image are mostly based on image processing and pattern recognition. Using the image in Figure 2.1 as a running example, the various standard operations, techniques, methods and tools used in the image analysis are introduced in this section.

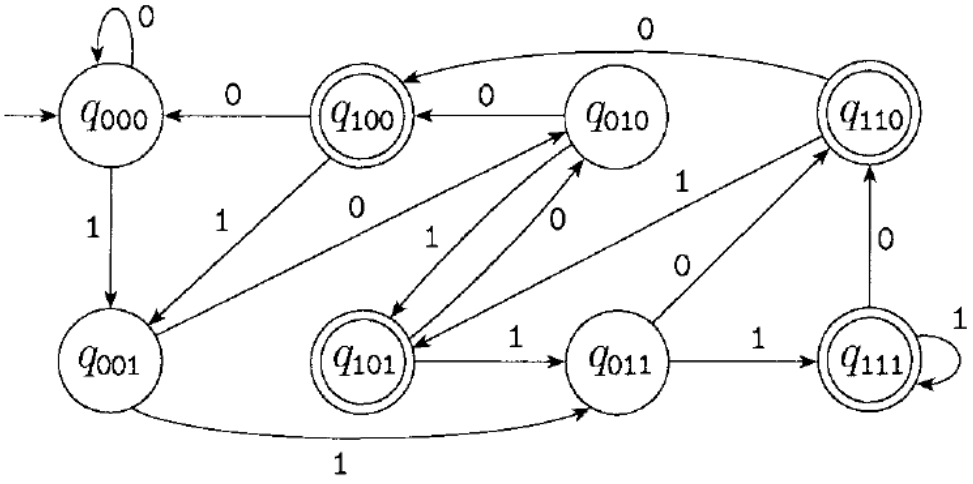


Figure 2.1: FSA diagram (original diagram from [138]).

2.1.1 Image acquisition and digitization

Digitization of a printed image is typically carried out using a 2D-input device such as a flatbed scanner. Camera hardware could also be used, provided uniform lighting exposure of the object can be obtained. Key issues at this stage include ensuring a good quality scanned image from the original print document, and using

a decent resolution above 200dpi to ensure details from the line drawing are not lost due to a low-resolution scan.

Scanning at higher resolutions, such as 600dpi, increases image size. The direct impact of this on a graphics recognition system is that more pixels need to be processed than if the scan was acquired at a lower resolution. Since most of the additional pixels are simply redundant, additional overhead in terms of processing is introduced. However, the skeletonization process (mentioned in later sections) will ultimately reduce the image to the barest foreground representation.

The possibility of missing some part of an image is higher in drawings with thin lines. On the other hand, artifacts from scanning from a paper medium could also appear in the digital image, thereby constituting noise. Just as human sight functions better in clear environments, a cleaner image may yield better results at the lower levels of the recognition system that deals with the manipulation of pixels.

Images may be scanned in colour, greyscale, or black and white. For FSA diagrams, we assume that images are scanned in greyscale. Greyscale images can easily be converted to black and white images (also called binary images) (see Section 2.1.5 for more detail). Our system converts the scanned FSA diagram to a binary image for further processing.

2.1.2 Raster representation

Digitization divides an image into a fine rectangular grid. Each cell on the grid is called a pixel (picture element) and each pixel contains the light intensity value(s) for that point of the image [86]; as well as *alpha* in some graphics systems and RGB (red, green, blue) values in systems based on the RGB model. The coordinate system for the display and manipulation of such raster images starts at the upper top left of the grid. In image processing and analysis these pixels can be compared, distinguished, grouped or removed in order to obtain the information depicted in an image. Figure 2.2 shows the standard pixel coordinate scheme.

2.1.3 Pixel neighbourhood

In raster images, analysis is carried out by examining pixel values and pixel patterns. For a particular pixel, the surrounding pixels touching it form its neighbourhood. The neighbourhood could be considered to be made up of a group of four pixels as illustrated in Figure 2.3a or eight pixels as illustrated in Figure 2.3b [135]. The 4-connected neighbourhood of a pixel location (x, y) can be defined as the set of pixels $\{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$. The 8-connected neighbourhood is the set of pixels $\{(x+i, y+j) | -1 \leq i,\ j \leq 1\}$, except for the case $i = j = 0$ [56].

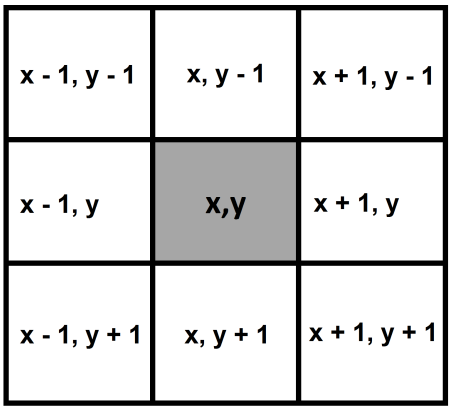
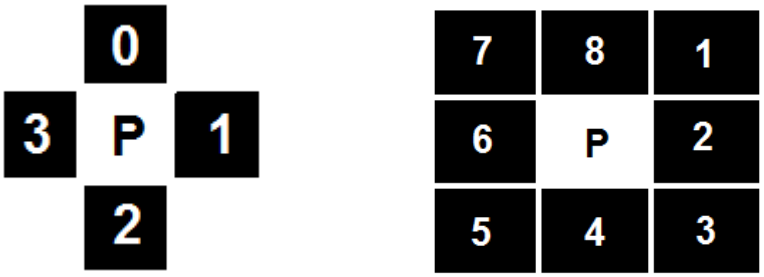


Figure 2.2: The pixel coordinates around a pixel p at location *(x,y)*.



(a) The 4-connected neighbourhood of pixel p.

(b) The 8-connected neighbourhood of pixel p.

Figure 2.3: Connectedness of a pixel p.

2.1.4 Mathematical morphology

Morphology concerns shapes. Image morphology applies special mathematical set operations on images. With the application of morphological operations, a new image is obtained. The new image results from either removing pixels from, or adding pixels to, the original image. The structuring element used in morphological operations is designed to analyse pixels of the original image. It is a relatively small image containing values used for analysing pixel environments in another image, and determines how the area under consideration in the target image should be affected by a morphological operation. The structuring element is moved across the image to be processed, visiting each pixel in turn. The structuring element represents a shape, has an arbitrary structure, and could be any size [135]. Common shapes for structuring elements are rectangle, square, diamond, and circle.

The morphological approach is a natural option in analysis relating to shape or form [72]. While morphological operations are usually used as part of the image analysis workflow, the techniques are versatile with some entire recognition systems,

such as MUSER [114], built mainly on mathematical morphology.

Basic morphological operations include erosion, dilation, opening, and closing. Erosion removes unwanted pixels from the image, and it is often used in disconnecting bridge points formed by the unexpected linking of connected components or removing certain parts of an image (for example, see Figure 2.4). Erosion reduces the size of an image region. Dilation on the other hand reinforces structures in an image; it fills gaps in regions and thereby expands structures in the image (for example, see Figure 2.5).

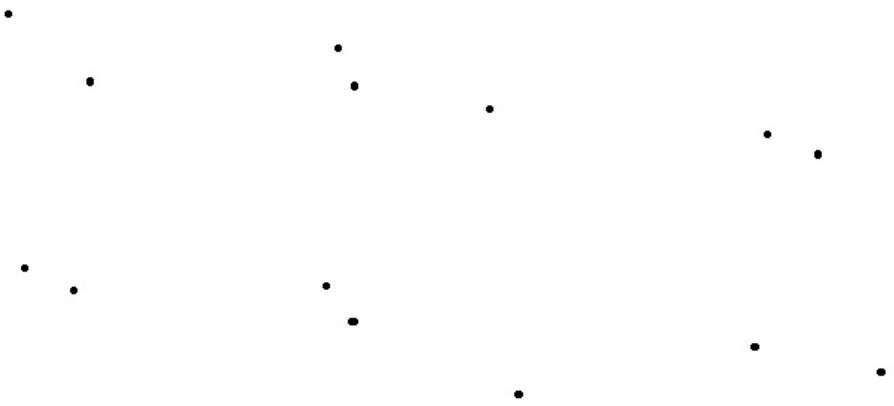


Figure 2.4: Erosion of an FSA diagram using a 5×5 circle structuring element. The dots are arrowheads of directed lines in the original image.

Opening involves carrying out a process of erosion and thereafter a dilation process. The operation gets rid of small portions of regions which may have extended into the background. The result of opening is that boundaries are smoothed, narrow isthmuses (links between larger areas) broken, and small noise regions eliminated [86].

Closing reverses the order by carrying out dilation and then following with an erosion operation. It results in closing up of the tiny gaps and holes in a region and eliminating 'bays' along the boundary [135].

Thinning (skeletonization), a more advanced morphological operation, is of importance in the analysis of document images and in character recognition. The skeletonization process is further discussed in Section 2.1.8.

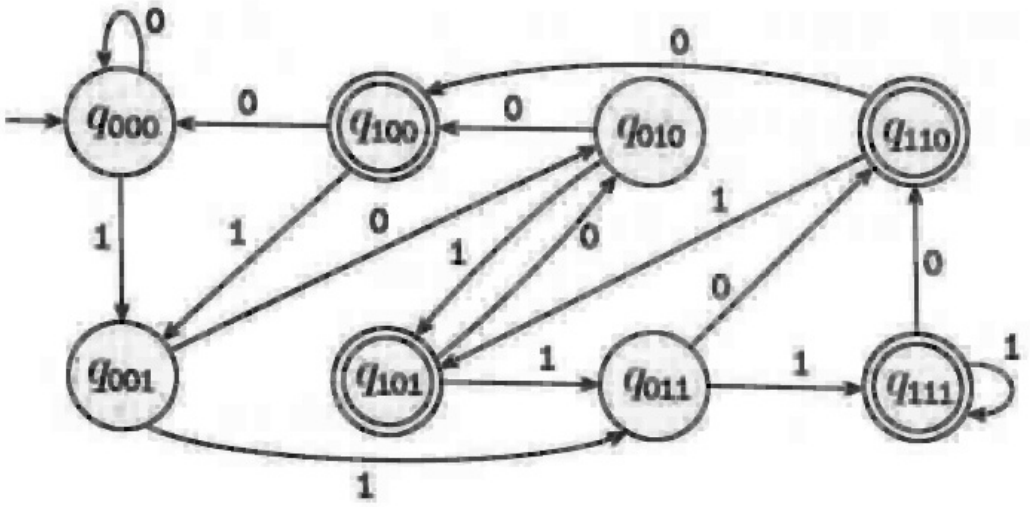


Figure 2.5: Dilation of an FSA diagram using a 3×3 structuring element.

2.1.5 Noise and noise reduction

Noise is unwanted signals inadvertently introduced into a digitized image. Noise-free images are an exception in image processing, and therefore several image processing techniques and algorithms for noise reduction exist. While noise could sometimes be invisible to the human eye, it is always present in images. The presence of noise is a potential complication in image analysis processes as noise pixels may end up being taken as valid information to be recognized, or valid image pixels eliminated as noise patterns. The FSA diagram in Figures 2.6 and 2.7 shows noise as grey smudges in the circles and around some arcs.

One class of noise in images is salt and pepper noise, which manifests as isolated regions of foreground pixels or background pixels. It also appears as rough edges of graphic components, as can be observed in Figure 2.7. The filling process is a technique for handling this type of noise, and involves covering the errant region with the pattern of surrounding pixels.

If the noise area covers multiple pixels, the O'Gorman kFill algorithm described in [86] could be used to eliminate the noise. Morphological operations (described in Section 2.1.4) are also used to remove noise pixels [86].

In our application, thresholding (the conversion from greyscale pixels to single bit black or white based on a predefined threshold) removed a large percentage of the noise found in poor diagram images (see Figures 2.8 and 2.9). Since global thresholding is a cheap operation computationally, we found this to be preferable to other noise reduction techniques. Furthermore, combining thresholding with size filtering produced suitable images for subsequent analysis, as the example image in Figure 2.10 shows.

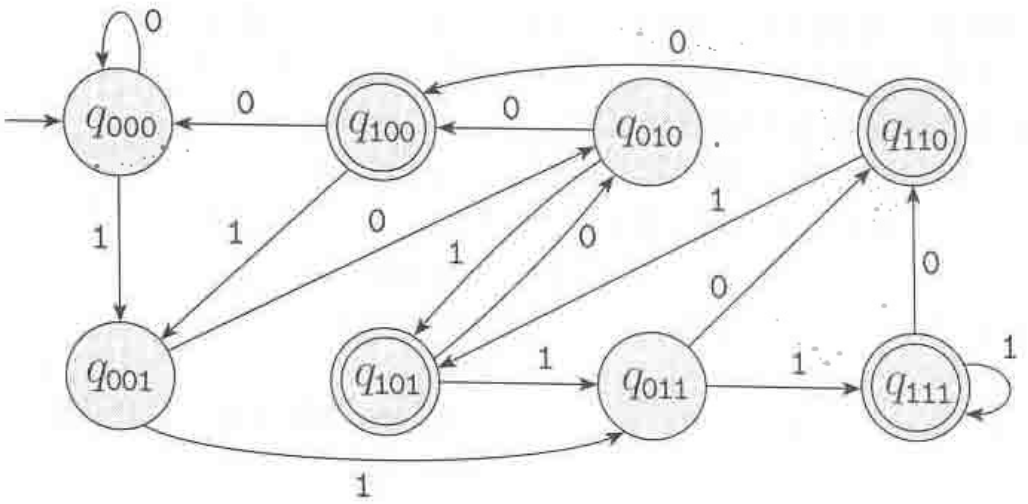


Figure 2.6: The noise-affected FSA diagram image before thresholding.

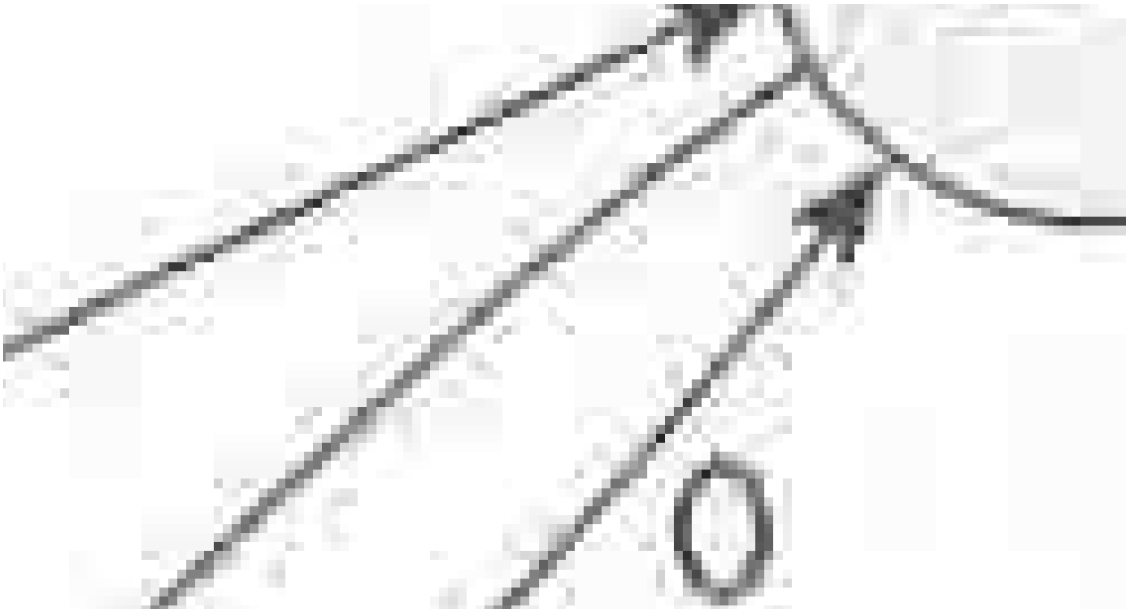


Figure 2.7: A segment of the noise-affected diagram image before thresholding.

Signal enhancement is a technique used in restoring missing parts of the information in a document image. Although it is similar to noise reduction, it uses domain knowledge to reconstruct missing parts of the image [86].

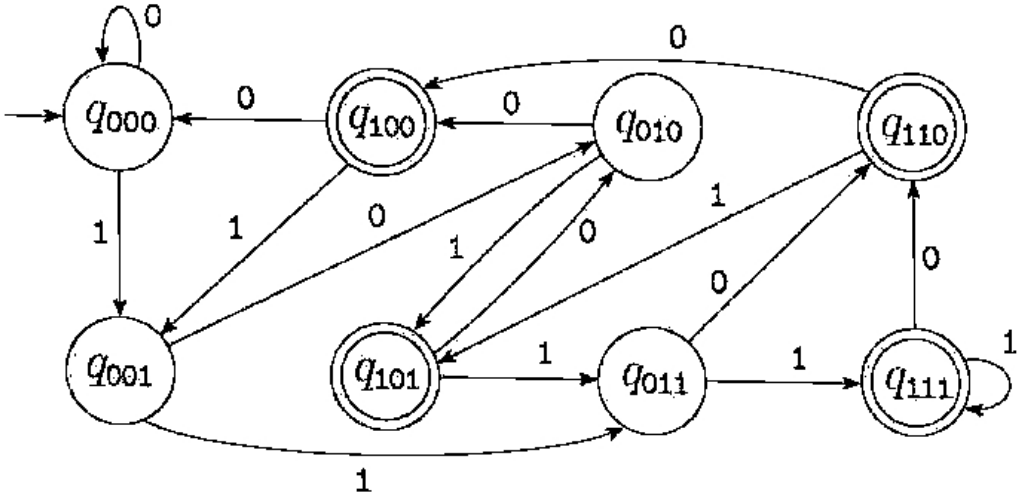


Figure 2.8: Diagram image in Figure 2.6 after thresholding.

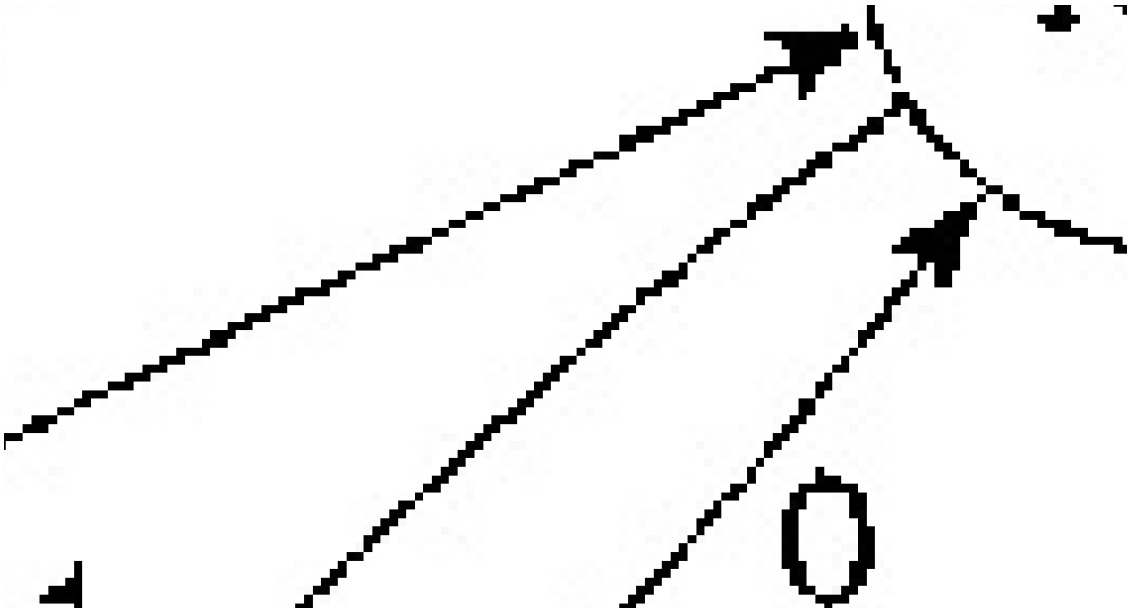


Figure 2.9: Zoomed segment of the noise-affected diagram image after thresholding.

2.1.6 Image segmentation

The human visual system has the ability to easily identify different objects and parts of an image or a scene. The variations among object features in an image, highlighted by their visual attributes such as colour, size, and orientation, are key to arriving at a decision about the different objects that constitute the image. Diagrams communicate by varying the arrangement of a distinct and recognizable

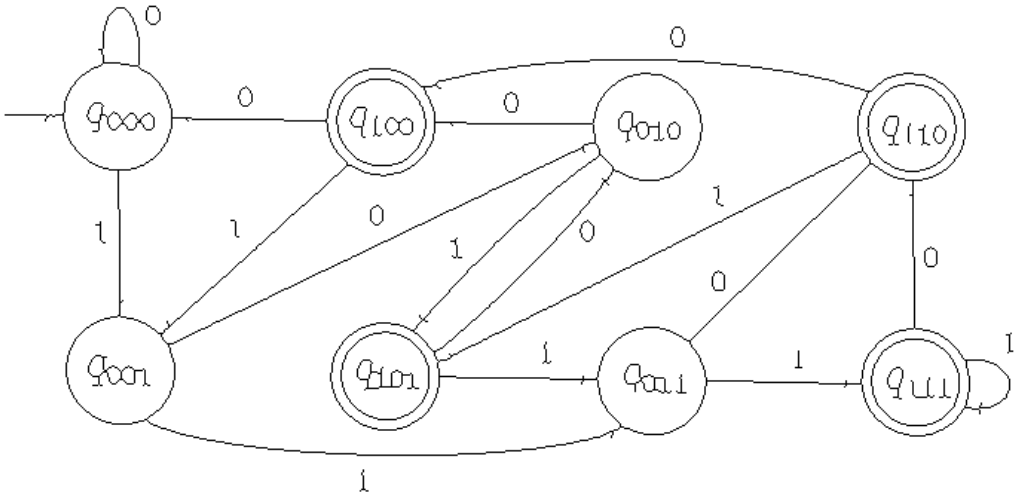


Figure 2.10: The FSA diagram after thresholding and thinning processes.

set of visual and non-visual elements [134].

In graphics recognition, algorithms are needed to assist in identifying objects of interest and for recognizing patterns and the different sub-images in the image. This is important since not much meaning can be directly extracted from a diagram image with all elements fused into a single connected component. The partitioning of an image into homogeneous regions is known as segmentation, and is usually application-based.

Segmentation occurs on two levels in document processing. Documents containing both text and graphics are first segmented to separate the text, and the graphics into different layers. A subsequent segmentation process then breaks down the graphics part of the document into individual components [86]. The type of segmentation required is determined by the particular application, and the type of graphics. From the various document analysis systems described in [46, 47, 96, 130, 131], it is observed that map documents require different segmentation to architectural drawings, technical drawings, and comics; within map images, different map types have differing segmentation needs too. In FSA diagram recognition, text-graphics segmentation and symbol segmentation is required. Segmentation is discussed further in later sections.

2.1.7 Vectorization

Although raster images contain light intensity values only, it is possible to obtain primitive objects such as curves and lines from the collection of pixels, and represent such objects using vectors. For instance, vectors are most appropriate for the representation of lines. Groups of pixels constituting a linear structure can be

replaced by a single vector.

Primitives are structural features in the image. Figure 2.11 shows a number of primitives found in diagrams. In vectorized images, instead of pixels, vectors and their coordinates represent image primitives, providing a higher level of abstraction than the pixel representation. The raster to vector conversion process is referred to as vectorization, and some known algorithms for vectorizing graphics documents can be found in [44, 116, 144].

Figure 2.11: Some primitives used in diagrams: dashed line, continuous line, curved line, and arrowheads.

For the interpretation of architectural, mechanical, and such technical line drawings, the vectorization process is essential for the diagram recognition process. However, for FSA diagram interpretation, vectorization is not compulsory. This is because the length, width, type and dimensions of lines do not form part of FSA semantics, and therefore obtaining them is not important.

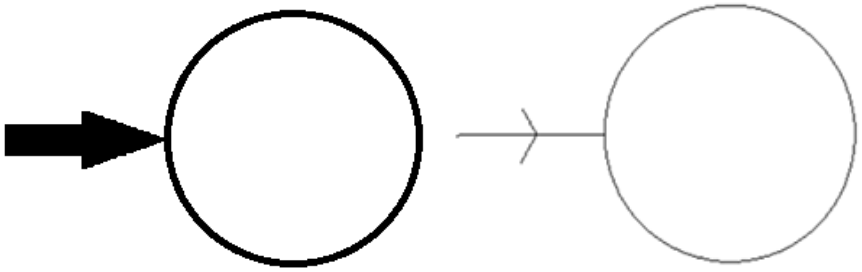
2.1.8 Skeletonization

The fundamental essence of skeletonization is the removal of redundant foreground pixels from a pattern to allow easier representation, processing, or analysis of the resulting form of the pattern called a skeleton [86]. The skeleton preserves the essential shape of the original pattern using the minimum number of pixels possible, while maintaining the connectivity of the pattern. A thinning operator (a morphological operator) is usually applied on a binary image to obtain the skeleton.

In Figure 2.12a an object is shown, with its corresponding skeleton in Figure 2.12b. A visual examination of these images reveals that they clearly represent the same concept, as the border thickness does not alter the perception of the fundamental shape.

This section is in no way a total coverage of image processing techniques used in diagram recognition. However, it is observed that for practical recognition of FSA diagram images, the various operations discussed produce adequate results for the automatic interpretation of these diagrams, as illustrated by our experimentation results in Chapter 6.

Each of the techniques in this section have far more interesting aspects to their application than what is covered here. The discussed techniques and several others,



(a) Multi-pixel thick. (b) Skeleton form.

Figure 2.12: A start symbol and its corresponding skeleton. The essential form remains the same, even though the line thickness differs.

including algorithms for practical image processing applications, are treated in detail in [122]. Other texts with an extensive coverage of the theory and applications of image processing in general include [42, 140].

2.2 Diagram recognition

Document Image Analysis (DIA) is a well-established scientific field [63]. Text capture and recognition technology termed Optical Character Recognition (OCR) are ubiquitous with many commercial and open source applications such as Tesseract [139]. The graphics recognition subarea of DIA has witnessed efforts into automatic recognition of various diagram types across diverse fields. For example, the recognition of architectural drawings [47, 100], engineering drawings [46, 49, 116, 150], maps [49], logic diagrams [89], charts [162], technical diagrams [76], electrical schematics [13], and line drawings in general [84] have been investigated.

In this section, diagram recognition systems are reviewed based on the diagram notation that the system targets, the processes involved, and the level of interpretation derivable from the system. The first criterium was chosen because the diagram recognition process is still mostly domain-dependent [14, 86], with recognition systems often depending heavily on heuristics based on the diagram notation. The third criterium to be studied is due to the fact that diagram recognition takes place at different levels, and recognition can therefore be characterized by whether the interest is the low-level or high-level of recognition [17, 126].

2.2.1 Diagram recognition processes and levels

Like any other complex systems, a diagram recognition system entails multiple processes. Different solutions are needed to solve the various sub-problems of the overall recognition task. The level of recognition targeted by a system determines the phases required.

Traditionally the recognition process can be broadly classified into two main phases; the low-level processing, and the high-level processing phases. The low-level phase is usually for the image acquisition, pre-processing, vectorization, and symbol recognition processes.

The analysis of the complete diagram representation for understanding, a process sometimes referred to as diagram interpretation, takes place in the high-level recognition phase. The two main phases are referred to as symbol recognition and symbol-arrangement analysis respectively in [14].

Ablameyko proposed a five-phase system for line drawings involving scanning, raster to vector transformation, entity extraction, scene formation and 3D reconstruction [3]. Pre-processing operations are merged into the first stage.

Blostein grouped the processes in diagram recognition systems to early processing, segmentation, symbol-recognition, identification of spatial relationships among symbols, identification of logical relationships among symbols, and the construction of meaning [14].

Kanungo et al., in their survey of engineering drawing understanding systems [82], describe three levels at which a recognition system could operate ranging from basic, to syntactic understanding capabilities, and then semantic-level understanding. Recognition phases are further categorized into lexical, syntactic and semantic phases. The lexical phase carries out the recognition of diagram primitives, extracting basic elements of the drawing; the syntactic phase uses a grammar to check the correctness of the recognized drawing with respect to the syntax rules of the particular diagram notation, and the semantic phase checks whether the recognized form represents a feasible object.

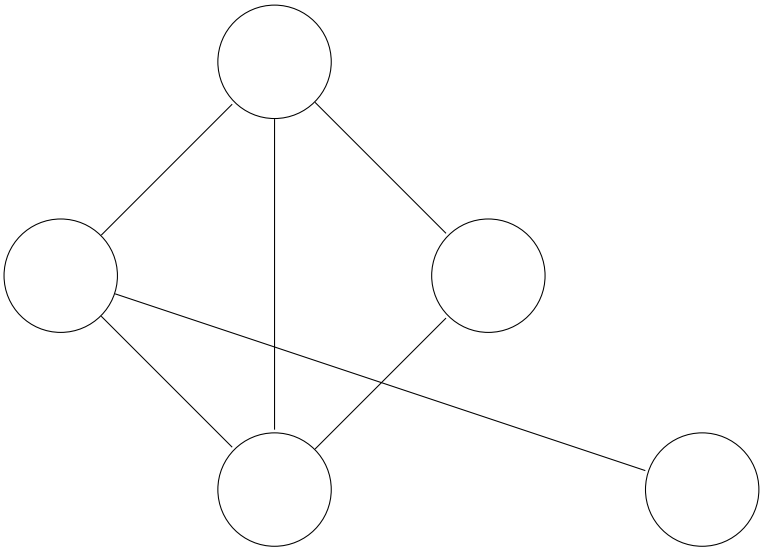
Graphics recognition was considered to involve two levels, the syntactic and semantic levels in [45], with a note that graphics recognition effort has mostly been at the syntactic level, and proposing that a distinction be made between syntactic and semantic graphics recognition.

The discussion in this section holds for interpreting various diagram types encountered in Science, Technology, Engineering and Mathematics (STEM) education. Although there can be significant variations in the notation of some diagrams in this group, the organization of a recognition system for most diagrams will generally fall under one of the schemes mentioned earlier in this section.

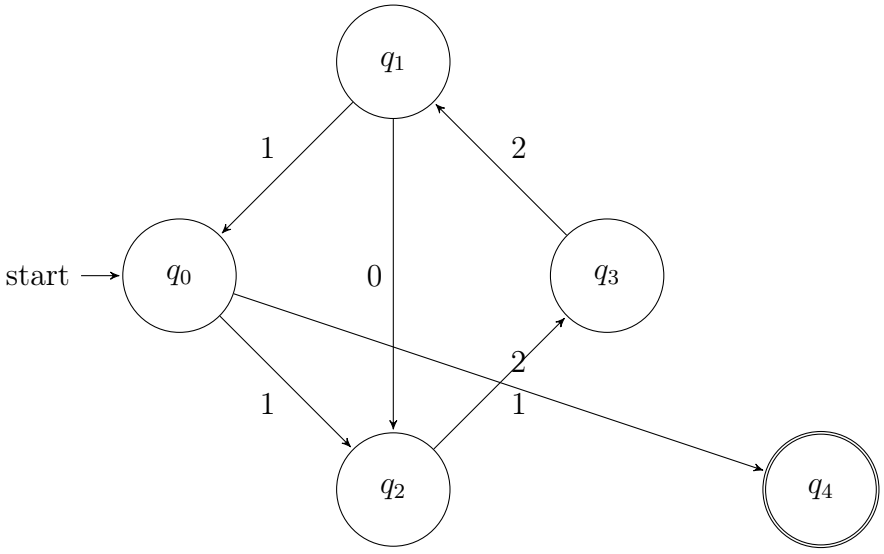
2.3 Existing diagram recognition systems

Several diagram recognition systems exist in literature. Since the visual structure of a diagram often influences the strategies used in developing its recognition system, we consider existing recognition systems built for diagrams that are structurally similar to FSA diagrams, or share common visual characteristics with the notation. FSA diagrams consist of simple symbols with interconnecting lines as the major

pattern for encoding information. A graph diagram is shown in Figure 2.13a and an FSA diagram in Figure 2.13b. The underlying commonly found primitives in such drawings are identified in Figure 2.11, while the symbols used in FSA diagrams specifically are shown in Figure 2.14.



(a) A graph diagram.

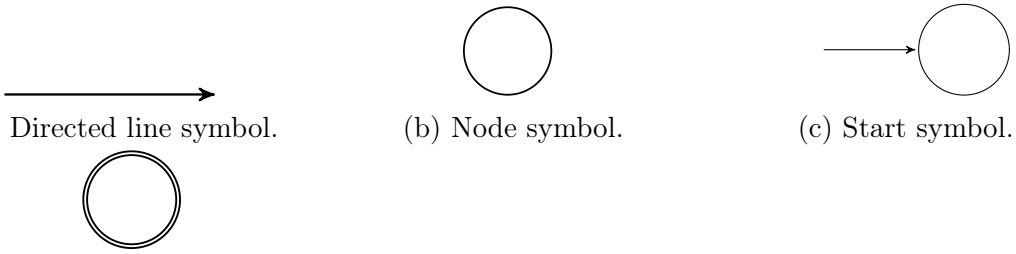


(b) Another dialect of graphs (FSA diagram).

Figure 2.13: Graph diagrams.

Restricting the scope to similar diagram notations certainly limits the works which could have been considered, but the major dissimilarity of structural, syntactic, and semantic organization between node-edge diagrams and the other diagram

types justifies their non-inclusion. For example, a flowchart is structurally closer to an FSA diagram than a musical score image, and the strategies used in interpreting the two diagram types (flowchart and musical scores) differ.



(a) Directed line symbol. (b) Node symbol. (c) Start symbol.

(d) Accepting state symbol.

Figure 2.14: Graphic elements of an FSA Diagram.

Diagrams having visual structures similar to FSA diagrams include conceptual diagrams, flowcharts, graphs, and schematic diagrams. We proceed to consider diagram recognition systems that have been developed for these diagram types.

2.3.1 Conceptual diagram recognition

A conceptual diagram is described in [51] as a systematic description of abstract concepts using predefined category boxes which have specific relationships between them; the diagram is typically based on a model or theory. See Figure 2.15 for an example of a conceptual diagram.

Conceptual diagrams are notably different from other node-link diagrams, as the notation is naturally ambiguous. Lines, text, and the nodes may be used to represent unrelated concepts in the diagram. For instance, lines are used to symbolize a connection relationship (like in node-link diagrams), and yet they can also be used to represent grouping, or the division of a group of concepts.

The interpretation of a conceptual diagram does not stem from any rigid predefined rules. In a way, the syntax of the notation is not established [155]. In conceptual diagrams, the interpretation of an instance of a diagram element is based on how it is currently used. The logical structure of a conceptual diagram is therefore directly derived from the physical structure of the diagram.

The conceptual diagram interpretation method described in [155] involves the extraction of physical objects, physical relations, logical objects, and the logical relations in the diagram. How the logical structure is obtained, is of more interest to us. Two processes are used in order to get the logical structure: hypothesis generation and hypothesis verification.

Hypothesis generation examines each element locally and assigns possible interpretations. All possible interpretations that can be made for a diagram element are extracted. Three categories of hypotheses are made: hypotheses of logical objects, hypotheses of logical relations, and hypotheses of labels. These hypotheses cover the range of possible interpretations which may be given to that element instance.

The hypothesis verification step then applies a set of constraints to filter generated hypotheses. The constraints examine whether a hypothesis case correctly satisfies all the conditions for the interpretation assigned to it. If it does not, it is rejected. This is important since not much meaning can be directly extracted from a diagram image with all elements fused into a single connected component.

The result of this approach to the semantic interpretation of diagrams showed [155] that only a third of wrong hypotheses were detected (a majority of the wrong hypotheses were not rejected), although all correct hypotheses were preserved. This is a notable limitation among others reported by the author, and it reflects one of the potential problems involved in attempting the semantic interpretation of diagrams using this approach.

Since FSA diagrams have a strict semantic interpretation, we will be able to avoid such wrongly identified hypotheses in our system (see Chapter 5).

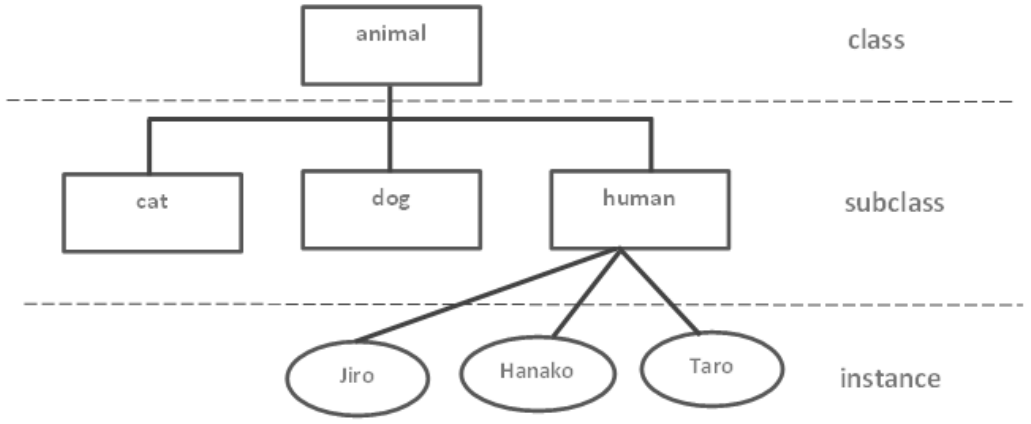


Figure 2.15: A conceptual diagram. Adapted from [155].

2.3.2 Flowchart recognition

Patent document analysis involves the recognition of diagrams included in patent documents. Rusinol et al. [132] developed a system which extracts structural in-

formation from flowcharts in patent documents and describes the structure in a predefined text format.

For the task, two approaches were proposed; one approach for pixel-based diagrams and another for a diagram which has been converted to a vectorial representation. Their modular architecture consists of text-graphics separation, an OCR engine, and node-edge segmentation modules. The pixel-based version of the system is of interest.

Text-graphics separation is carried out in the system by examining features such as orientation, size, and the height and width ratios of connected components in the image. A number of adaptive thresholds are then used to decide whether the connected component corresponds to a graphical element or a textual element; with the result going into text, graphics, or undetermined layers. We follow a similar approach in our system.

To segment node symbols from lines, connected components representing areas within closed borders of lines are examined. Examples of such areas are illustrated in Figure 2.16 and marked *CC1* to *CC6*. Small and oversized connected components are filtered, while the remaining node candidates are analysed to determine which connected components are potential nodes, and which are formed by invalid loops between node-link-node connections (such as the shaded area *CC5* in Figure 2.16).

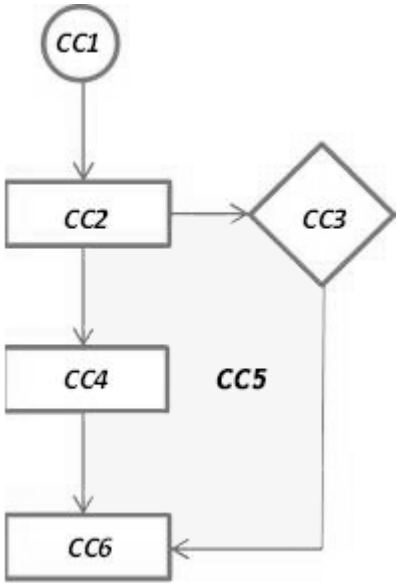


Figure 2.16: Connected components of white pixels in an example flowchart diagram segment are labelled. The region CC5 is an invalid loop.

Two features of the node candidate objects are used to discriminate them, namely, solidity and vertical symmetry. Objects with solidity lower than a threshold are dismissed, since nodes tend to be convex. The vertical symmetry is computed

as the ratio of the sum of the pixels on the left, to those on the right part of the connected components. Objects below a symmetry threshold are also dismissed, because nodes tend to be vertically symmetric. To classify the nodes into shape families used in flowchart diagrams, the blurred shape model descriptor and geometric moments are used.

Detecting which line connects a pair of nodes is accomplished in the system by finding out which set of lines makes two separate connected components of nodes merge into a single component.

The interpretation level of the system is limited to the structural form of the diagram; neither the syntax nor the semantics of flowchart notation were explicitly applied in the interpretation.

The interpretation of flowcharts is also reported in the system of Bunke et al. described in [21]. They assume recognition of primitive levels have been carried out and only perform recognition of a flowchart diagram at the diagram levels using an attributed programmed graph grammar. Another multi-domain recognition system described in [157] uses rules and a matcher to recognize symbols and interpret simple drawings.

2.3.3 Graph diagram recognition

The work of Auer et al. in [6] is the closest recognition system to ours; it dealt with the recognition of traditional graph diagrams such as shown in Figure 2.13a. The motivation for their research is to extract the topology of a graph for use in a graph drawing environment. The input to the system is a graph diagram.

Their system requires four phases, namely, pre-processing, segmentation, topology recognition, and post-processing. The topology recognition phase creates a skeleton of the image and labels the pixels as vertex, edge or crossing. The problem of crossing edges, which is common in non-planar graphs, was handled by following lines in the manner that a human being will visually follow the lines, tracking it from one end to another along the visual path.

In their system, nodes were assumed to be filled circles. For nodes which are depicted by outlines alone, they suggested using the Hough transform to detect the node (circle) and after detection, applying a filling process to fill the inner portions of the node. However, in our experience, such an operation is complex and its success in practical applications require additional steps to confirm the results obtained from the Hough transform process (in particular, the input parameters for the Hough transform depend on the characteristics of the graph used as input).

The recognition system detects the nodes and their interconnections, returning an interpretation of the topology of the diagram only.

2.3.4 Schematic diagram analysis

A schematic diagram (see Figure 2.17) uses different symbols, connecting lines which connects symbols, and text annotations to create a simplified representation of a circuit. Schematic diagrams are characterized by symbols representing electrical components of the circuit and connecting lines representing conduction.

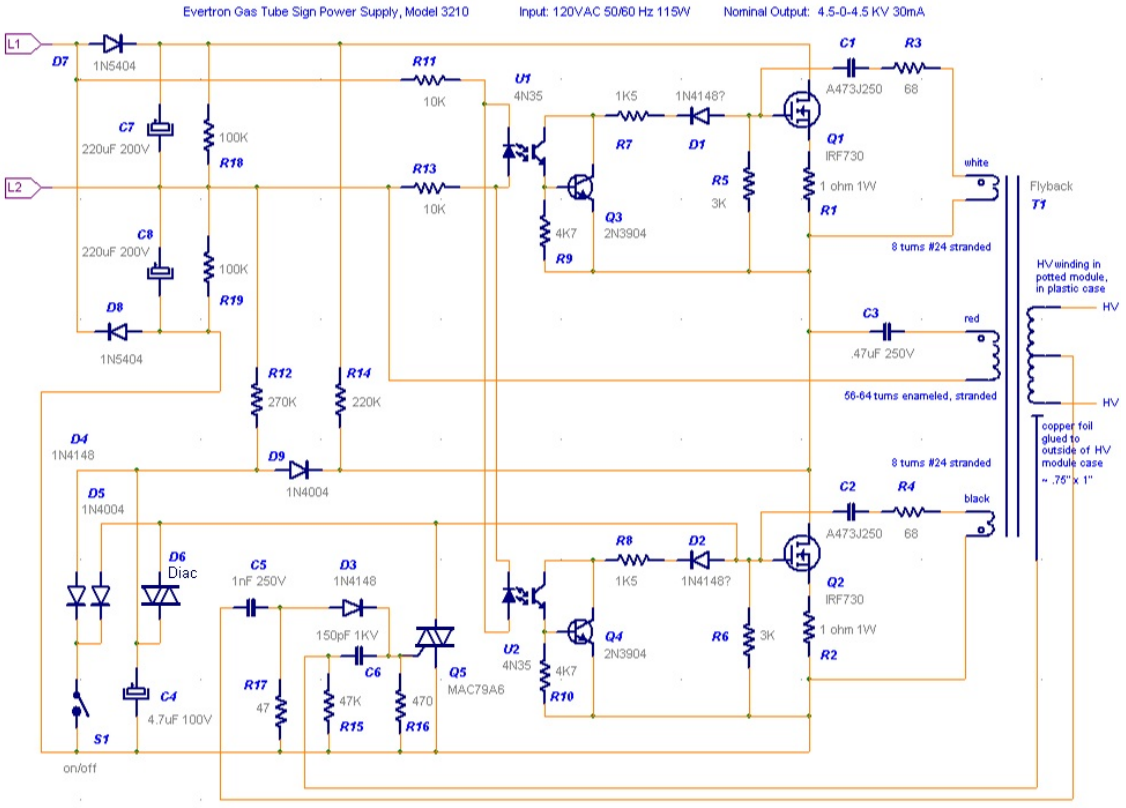


Figure 2.17: A schematic diagram. Original diagram from [67].

In [71], to solve the task of locating the symbols present in a schematic diagram, a hypothesis building and verification approach was used. Hypotheses of candidate areas which are more likely to contain a symbol are made, and supporting evidence sought from the diagram to support the decision. To make the hypothesis for a candidate area, the presence of endpoints of connecting lines are used as triggers.

To support the hypothesis, a text label or loop is sought around the candidate area. A loop is an enclosed area formed by a closed shape. Symbols in a circuit diagram such as the one representing an amplifier, is an example of a loop symbol. Loop-free symbols, such as for example a capacitor, do not contain loops. Once all symbols have been detected, the connecting lines are then identified.

On the whole, the recognition system is organized into low-level and high-level recognition phases. The low-level phase recognizes the primitives, while the high-level phase identifies the symbols using models of known symbols. The output of the recognition system is a localization and recognition of symbols and connecting lines present in the diagram.

The system shows one way by which recognition systems exploit prior knowledge of the structure of the diagram to assist the process; in this case, the use of labels and loops to locate symbols. Similar systems for the recognition of circuit diagrams include a hand drawn circuit diagram recognition system [48] and Bunke et al.'s system described in [21] for the recognition of schematics and and flowchart diagrams using attributed programmed graph grammars.

2.3.5 Multi-notational recognition systems

Graphics recognition systems are mostly domain-specific, because the recognition strategy is usually based on the form of the diagram notation and its semantics.

However, Kasturi et al. demonstrated a system that is designed to interpret various diagram notations [85]. They noted that a recognition system which generates a description of the symbols and their relative spatial placement from a raster image has many applications.

The proposed system is used to describe scanned graphics in terms of a set of known shape primitives (polygons, circles) and their interconnection. Designed to be comprehensive, the system is able to separate text from graphics, recognize graphical primitives, identify line segments, and describe the diagram via a connection list. Primitives are discovered using a pattern recognition approach on line segments, and loops are found by analysing the segments.

The system recognizes the primitives and basic shapes present in the diagram. The symbol recognition is limited to recognizing a preset group of known shapes which are mainly polygons and circles. All other loop-shaped element not matching those, are described as complex loops and broken into line segments for further analysis. A similar system for the recognition of multiple types of engineering drawings is described in [157].

Getting a complete interpretation (including the semantics) of a diagram requires the use of domain knowledge to correctly interpret the symbol-connection descriptions extracted from the diagram. This is because each symbol in a diagram is linked to certain semantic concepts which are usually specific to a domain. On its own, the system is not capable of such a level of interpretation. Subsequently, an additional process will be required to provide semantic interpretation capabilities for the notation or domain of interest.

Towards solving the challenges of analysing multi-dimensional representations such as diagrams, graph grammars were extended by Bunke in [21], and applied to the interpretation of schematic diagrams. The recognition of nodes and connections

in flowcharts and circuit diagrams using attributed programmed graph grammars was demonstrated.

The use of graph grammars for the analysis of diagrams is not new, but the manner in which the grammar is used in the case of the system is unique. Instead of a parser operation applying the grammar, the grammar is used to generate an output graph by explicitly programming the order of application of the productions. The attributed programmed graph grammar uses applicability predicates which enforces constraints on the left-hand side of the production in order for it to be replaced by the right-hand side graph. In that way, the well known complexity involved in parsing graph grammars is bypassed.

The grammar was applied to the automatic extraction of a diagram's description in terms of the symbols present and their interconnections, given a schematic diagram represented as a collection of line segments (they assume these are already extracted from the image by some other process).

The original diagram made up of line segments is first given an intermediate representation (a graph representation). In this graph, the nodes corresponds to vertices in the diagram and the edges corresponds to connecting lines; this is the input graph. Similarly, the interpretation result is also represented by an output graph. In the output graph, nodes represent the symbols in the diagram and the edges represent the connection lines in the diagram. The input and output graphs are augmented by attributes. If the input graph can be successfully transformed into an output graph, the visual sentence (represented by the input graph) is a valid diagram instance of the notation described by the grammar.

The advantage of this approach compared to a traditional graph grammar system is the fact that parsing is unnecessary. But this has a disadvantage, in that the absence of a parser results in not being able to have a hierarchical overview of the organization of diagram elements as would have been possible if a parser was involved. Also, the need to have an input graph representation of the original image and a set of programmed productions appropriate for transforming the input graph makes the approach somewhat complex.

We now briefly consider available methods for the separation of text and graphics.

2.4 Text-graphics separation

Diagrams often contain character symbols and alphanumeric character strings. In FSA diagrams, these occur as labels and annotations. Text-graphics separation discriminates character strings' pixels from graphics pixels. The output of this phase are separate text and graphics layers. The text layer can be processed by a standard OCR module, which may be a part of the system or an entirely separate OCR application.

For the purpose of semantics analysis of diagrams, all meaningful elements of the diagram image with respect to the notation under consideration, need to be extracted and considered. It is almost impossible to obtain the semantics of FSA diagrams without the correct detection of state and transition labels. The text-graphics separation phase is therefore critical to the success of the analysis.

Text-graphics separation methods are usually application-dependent [58]. The nature of the parent document, whether text-rich, for instance a newspaper page, or graphics-rich as in the case of FSA diagrams, also matters. Therefore, algorithms for text segmentation of regular documents may not be able to effectively extract text in graphics-rich documents.

A characteristic of the annotations in FSA diagram is their sparseness, often consisting of only a few characters in length. The sparseness and the possible presence of subscripts, make text detection challenging in these diagrams; a valid single-character label could be erroneously classified as noise, or missed altogether. More challenging is the case of characters touching lines or other graphics.

Well-known algorithms for text-graphics separation include [58]. However, this algorithm requires a minimum string length of three characters. In FSA diagrams, labels often consist of only one or two characters making the algorithm unsuitable for use in FSA diagrams in its original implementation.

For engineering drawings, the method described in [101] is designed to extract text of any orientation, character length, and type – whether western or other character family, and is robust as concerns text touching graphics. Although many parameters and thresholds are involved, the fundamental techniques could be adapted to detecting and extracting character regions in FSA diagrams. Text-graphics separation is further examined in Section 3.3.1.

2.5 Shape and symbol recognition in images

From a holistic overview of diagram recognition systems, it is clear that diagram recognition concerns isolating and identifying the elements in an image. Aside from text and lines, the other visual elements in diagrams are symbols, and therefore a cursory consideration of shape description and recognition processes in images is necessary.

In most of the recognition systems discussed before, it can be observed that a key part of the recognition system is the detection and identification of the symbols present in the diagram. Sometimes these symbols are plain geometric shapes, and for those which are not, the process of symbol recognition can be taken as a particular instance of shape recognition [148]. FSA diagram symbols are basic geometric shapes, and therefore our focus is set on shape recognition methods suitable for the analysis of geometric shapes.

Various approaches to symbol recognition are known [4, 27, 32, 145]. The reader may consult [98] for a classification based on applications and techniques.

To determine the presence of a shape in the image, a shape template may be required for matching occurrences of the shape in the image [95]. A shape can be identified using its shape representation information or the shape description information. Shape recognition requires that the shape description contains enough information to distinguish the shape of interest from other shapes that are present in the image, even if the shape is present in a different scale, position, or orientation than that which is described [110]. Therefore, the description used must be invariant to these possible variations in the appearance of the shape in an image. This section examines binary shape analysis.

2.5.1 Shape representation and shape description

The input to shape analysis algorithms are binary images obtained from a prior segmentation process [99]. Thereafter, shape analysis may be carried out using shape descriptors or shape representations.

Shape representation and shape description have received much attention [159]. Our interest is 2D shape representation and description because FSA diagrams do not depict 3D views or objects. Shape representation refers to the group of techniques developed to capture the properties of shapes for further analysis. These methods produce a non-numeric representation of the original shape, maintaining important characteristics of the shape which can be used for their analysis, while shape description methods results in numeric definitions for characterizing the shape (shape description vector).

With several shape representation techniques available to choose from for the shape recognition operation, the question therefore is which is most suitable. Invariance (rotation, translation, and scale), robustness to noise and minimal distortion, low computational complexity and application independence are some factors which should influence the choice of shape representation and description in an application [159].

We will not attempt an exhaustive survey of shape analysis techniques, but will rather focus on a specific review of approaches which have been used in applications for the recognition of geometric shapes. Several reviews and discussions about shape representation, shape description and shape detection covering their workings, nature, and descriptions can be found in [54, 95, 110, 121, 124, 159]. Detailed taxonomies of shape analysis, covering description and representation, can be found in [95, 123, 140].

We proceed to consider some options available for the task of characterizing shapes.

2.5.2 Simple shape descriptors

Some features of an object can be used to describe the shape. A feature is a specific measure that is computable from the values and coordinates of the pixels that make up the region [22]. Table 2.1 contains a selection of features which are relevant to determining the nature of a binary image object; although there are several features which can be used (see [120]), the selected features are sufficient to distinguish and characterize the various elements found in FSA diagrams, and were therefore highlighted.

Property	Description
Area	The area of the diagram element is the number of pixels that it consists of. The area A, can be computed as a summation of all pixels in the region.
Convex area	The convex area of an image region is the number of pixels within its convex hull boundaries.
Eccentricity	The eccentricity of an image region is the ratio of the major and minor axes of the region. The value is between *0* and *1*.
Equivalent diameter	This feature is the diameter of a circle with the same area as the image region. It is calculated as $EquivDiameter = \sqrt{(4 \times Area)/\pi}$.
Extent	Extent is the ratio of the region area to the bounding box area. The bounding box is the coordinates of the smallest rectangle which can enclose the image region.
Minor axis length	The length of the minor axis of the ellipse that has the same normalized second central moments as the region. This value is in pixels.
Solidity	The solidity of the image region is computed as $Area/ConvexArea$.

Table 2.1: Some binary image features [104].

Such measurements as the area, perimeter, moments (moments are discussed in Section 2.5.6), holes, convex hull, and enclosing rectangles, are object properties which could be used to analyse object shapes in binary images.

Holes are the empty regions formed by background pixels within an object's structure. In the study of an object's topology, the presence of holes and the number of holes can be used. For example, in terms of its structure, the visual representation **'8'** has two holes. This feature alone may not be sufficient enough to classify a shape.

The Euler number of an image object is the difference between the number of connected components and the number of holes in the object. The Euler number is invariant to scale, rotation, stretching, and translation changes of the image object. However, since the possibility of two different objects in a multi-object image having the same Euler number value exists, the Euler number alone cannot be relied on to differentiate objects in some instances.

The area and the perimeter covered by an object in an image can also be measured and analysed. Just as human beings use size to categorize and make deductions, the relative size of an object which is obtained by measuring the perimeter or area could be used to discriminate objects, especially when prior knowledge of object sizes in an image is known.

The convex hull is the area covered by the extent of the form of an object. It is the smallest convex region which would cover the region of the object. The convex hull in some cases is not rotation-invariant [140] and therefore this feature could miss detecting an object as a match.

2.5.3 Structural and syntactic techniques

The structural approach to encoding shapes represents the shape structure by patterns of a finite set of atomic components (primitives). These components are consequently encoded to represent the visual form of the object using a string or graph [95].

Syntactic methods can be considered part of structural pattern recognition methods [142]. The relationship between patterns and their sub-patterns forms the basis for these methods. From the boundary topology of the object, it is possible to obtain a representation based on its primitive elements (see Figure 2.18.)

The representation involves the primitives and a set of rules detailing what objects can be formed from these primitives and the arrangements forming those objects. An example of how primitive elements can be used to represent an image object is shown in Figure 2.18.

Grammars are useful in applications where the structure of the symbols (or objects) can be treated as a set of rules [98]. Formal grammars (usually graph grammars or its variants) are used for the structural analysis of symbols due to the non-linear nature of visual constructs. Graph grammars and their applications to images are discussed in [117].

Structural and syntactic approaches have been used at different levels of the diagram recognition problem. Syntactic methods have several applications in diagram recognition. Grammars have been used at the symbol recognition phase for shape representation, and have also been used in the overall recognition of a particular diagram instance; the use of grammar for a music score recognition system described in [40] is an example. A discussion of the advantages and the challenges of syntactic methods are discussed in [142].

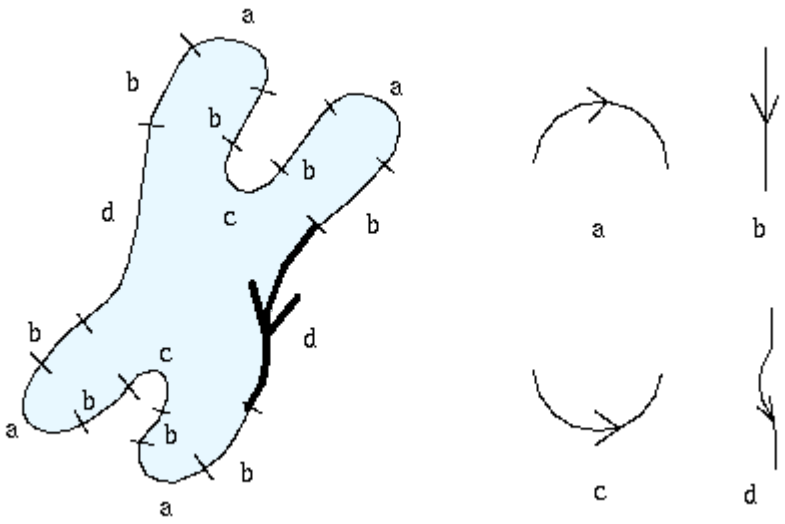


Figure 2.18: Using the structural technique to describe the boundary of a chromosome image sketch [8].

2.5.4 Chain codes

A chain code [60] is based on the contour of an object and it offers a structural report of the topology of the object. It is a set of directional codes obtained by traversing the boundary of an object in a raster image from a starting point, and recording changes in direction along the boundary of the object. Figure 2.19 marks out the path direction which may be encoded for the object in an image.

Each of the possible directions from a pixel point has a code, and the location of the next foreground pixel is determined and added to the set of codes for the object. Chain codes efficiently represent the contour of an object or curve. The chain code representation is suitable for syntactic pattern recognition.

The challenge with this method is that the chain code is sensitive to noise, arbitrary rotation, and scale perturbations [140].

2.5.5 The Hough transform

The Hough transform is a feature extraction method. While it is well known for detecting lines, it has also been used to detect curves, circles, and ellipses [120]. The Hough transform exploits the mathematical properties of lines and shapes, and uses parameters of the object to detect instances of its occurrence in an image. It therefore requires a parametric description of the shape sought.

Consider an image on which the Hough transform is to be applied. After thresholding and thinning, or edge detection (an image processing operation which computes edge vectors such as gradient), is carried out on the image, each foreground

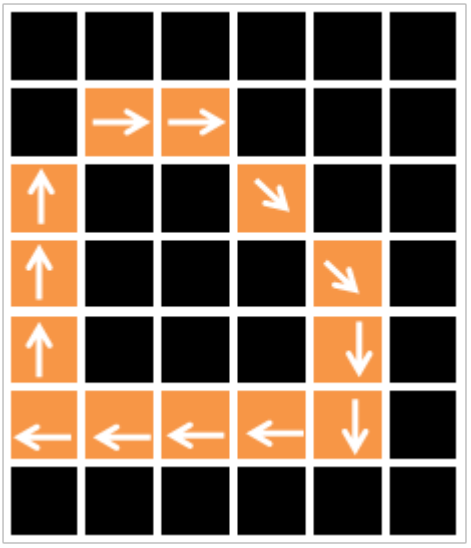


Figure 2.19: Chain codes path in a sample image.

pixel can be represented in a *parameter space*. The method seeks evidence of the existence of the line or circle in the image by examining the parameter space. For example, the equation for a line is:

$$y = mx + c \quad . \tag{2.5.1}$$

Each point on this line is represented by a line in parameter (m, c) space. All pixels forming the line in the (x, y) plane passes through a single point in (m, c) (parameter) space. The point of intersection of all these lines gives the values of the parameters m and c in the equation $y = mx + c$.

For vertical lines, the polar (ρ, θ) form of expressing a line is used. A line in the (x, y) plane can be represented by [120]:

$$\rho = x\cos(\theta) + y\sin(\theta) \quad . \tag{2.5.2}$$

From this polar form, values for m and c are:

$$c = \frac{\rho}{\sin(\theta)}, \text{ and } m = -\frac{1}{\tan(\theta)} \quad . \tag{2.5.3}$$

For detecting circles, the Hough transform for lines is extended by using the equation for a circle represented in parametric form. This equation is given as [120]:

$$x = x_0 + r\cos(\theta), \text{ and } y = y_0 + r\sin(\theta) \quad . \tag{2.5.4}$$

If the expected radius of the object is known, only two parameters x_0 and y_0 remain to be obtained. The computation required to detect the object is therefore reduced if the expected radius is known.

The use of the Hough transform for pattern recognition applications in image processing and computer vision can be grouped into two categories. The first approach uses parametric features of the shape to analyse an image, detect the peaks in the transform space and subsequently makes judgments to support the hypothesis of the presence of a shape. The second approach is the hierarchical approach, where the Hough space is interpreted for features which can be collected to form global structures [25]. The Hough transform and its various implementations and example applications are covered in-depth in [120].

The main advantage of the Hough transform is its relative insensitivity to noise and the ability to work without the need for a continuous contour description [110], and therefore missing segments of the line does not hinder the detection of an object.

The Hough transform was considered for detecting the circles and lines in FSA diagrams in our early experiments, but ultimately the technique was not used. The need to define the correct parameters in advance, coupled with the fact that different variations of curved lines cannot be ruled out in FSA diagrams, the computational costs of the approach, and the need for further processing before the objects of interest are correctly localized and identified, weighed negatively on the choice of applying it in a diagram recognition system.

2.5.6 Moments and moment invariants

Moments are scalar measures used to characterize functions. The application of the principles of moments to pattern recognition was introduced in [77]. In application to visual patterns, moments are statistical properties of a shape and yield a global description for the shape. Various systems of moments exist.

A pattern can be represented by its two-dimensional moments with respect to a pair of fixed axes. The two-dimensional Cartesian moment is associated with an order, starting from zero as the lowest order to higher orders. Using the higher order moments, descriptors which are invariant to scale, rotation and position can be obtained. For describing shapes which are fairly round, moments have proved valuable, but they have also been used for more complex applications such as airplane silhouette recognition [42].

The two-dimensional geometric moment of order $(p+q)$ of a function $f(x,y)$ is defined as:

$$M_{pq} = \int_{a_2}^{a_1} \int_{b_2}^{b_1} x^p \; y^q \; f(x,y) \; dxdy \tag{2.5.5}$$

where $p, q = 0, 1, 2, ..., \infty$ [97].

The sum $r = p + q$ is referred to as the *order of the moment* as defined in the preceding equation.

The zeroth order moment of the function $f(x, y)$ is given by:

$$M_{00} = \int_{a_2}^{a_1} \int_{b_2}^{b_1} f(x, y)\ dxdy \quad . \tag{2.5.6}$$

This value represents the total area when computed for a binary image.
The two first order moments are defined as follows:

$$M_{10} = \int_{a_2}^{a_1} \int_{b_2}^{b_1} x\ f(x, y)\ dxdy \tag{2.5.7}$$

and

$$M_{01} = \int_{a_2}^{a_1} \int_{b_2}^{b_1} y\ f(x, y)\ dxdy \quad . \tag{2.5.8}$$

The central moments of $f(x, y)$ are defined as:

$$\mu_p q = \int_{a_2}^{a_1} \int_{b_2}^{b_1} (x - \bar{x})^p (y - \bar{y})^q\ f(x, y)\ dxdy, \tag{2.5.9}$$

where $\bar{x}$ and $\bar{y}$ are defined as:

$$\bar{x} = \frac{M_{10}}{M_{00}}, \quad \bar{y} = \frac{M_{01}}{M_{00}} \quad . \tag{2.5.10}$$

Geometric properties of shapes such as area, center of mass, and orientation can be obtained from moments. For a binary image, the zeroth moment equals the area covered by the shape, the two first order moments represent the center of mass and the second order moments are used to determine orientation of the shape of interest [97]. Translation invariant features are computed using centralized moments of the second order; scale invariance is achieved by computing normalized central moments, and seven different rotation invariant moments are computed from these normalized moments. The method of moment invariants with which invariance to scale, position, orientation, and translation is achieved, is defined in [77].

2.6 Assistive technology related diagram recognition research

The diagram recognition systems discussed prior to this section have been driven by primary motivations outside assistive technology uses. While this may seem to paint a picture of lack of assistive technology based diagram recognition research,

this is not the case. Previous works exists in this area and some are subsequently discussed in Section 2.6.1.

Our choice to mainly report work with their foundations in image processing, pattern recognition and graphics recognition is because those areas form the bedrock of diagram recognition regardless of the objective or application. Also the efforts, techniques and systems already existing in the aforementioned research areas need not be recreated.

Two sets of themes are distinctly observed in assistive technology driven diagram recognition literature. There are efforts targeted at automating tactile graphics generation, and another set focused on the re-presentation of graphical information for blind users.

2.6.1 Diagram recognition in assistive technology

One of the relatively early research in making paper-printed circuit diagrams accessible to blind people is the system described by Kawai et al. [87]. A camera image of the diagram is obtained, thinned, and singular points such as end points, branches, and crossings are determined. Lines are segmented and labeled, and circuit components and connections are recognized. These are displayed on special tactile hardware.

Zapirain et al. designed a system to transform electric schematics into a form that can be verbalized to a blind user [158]. Image processing techniques are used to obtain the information contained in the schematic diagram, and the information is then rendered in textual form.

Way studied the automatic production of tactile graphics [151] and developed a tool for the process. Although the research was not about diagram images but images in general, it should technically be able to generate tactile output of a diagram image.

It should be noted that the systems mentioned do not interpret or attempt to understand the information contained in the image, but they rather produce a version of the image suitable for hard copy tactile prints.

A recognition system for automatically recognizing printed beading design diagrams is introduced in [75]. The system uses template matching, and the recognized pattern is rendered as natural language to the user.

Notable and concerted efforts by a consortium of universities and government organizations to bridge the accessibility gap resulted in the TeDUB (Technical Drawings Understanding for the Blind) project which attempted to provide blind computer users with access to technical drawings across a number of 'formally defined domains' [59]. Electronic circuits, UML (Unified Modeling Language) diagrams, and architectural drawings are some of the diagrams treated, although the goal of the project was a system not limited to any diagram domain. The system accepts as input a bitmapped image or a file containing semantic information (the

diagram interpreter), and has an interface to make the information contained in the files available to a blind user via a diagram navigator.

2.6.2 Representation of diagrammatic information to blind people

Bennett studied how diagrams can be properly presented to blind users when using sound as the means of presentation of the diagrams [9]. The presentation of graphs representing electrical heating schematics, via computer generated sounds, was considered in the study.

A manual (completely non-electronic) technique of producing inexpensive representation of FSA diagrams to students is described in [18]. The existence of this work highlights a need for automatic diagram interpretation solutions to aid in teaching visually impaired students.

Audiograf [88] is a model that uses tactile exploration on a touch panel interface, and audible presentation of the diagram elements currently being touched by the users, to represent diagrams to a blind user.

The GSK tool described in [7] allows blind users to create, edit, and share graph diagrams using a combination of standard computer peripherals such as a mouse, keyboard, monitor and screen reader. A similar tool is PLUMB [23], with which a blind user utilize the tablet and a pen to interface and use graph diagrams.

Some other studies in the alternative presentation of graphical information include [5, 12, 28, 90, 127, 156].

2.7 Chapter conclusion

Low-level processing, shape detection and recognition are major aspects of diagram recognition. These have been considered in the foregoing sections with respect to the analysis of node-connection or network type diagrams. It is observed that there are various options when it comes to methods, techniques and algorithms for handling the various facets involved in solving the diagram recognition problem.

Chapter 3

Diagram image analysis

"An important class of visual languages are the so-called diagrammatic visual languages: each sentence is composed of graphical objects which have certain spatial relations among each other, and the interpretation of such a sentence is fully determined by these visible aspects" [128].

Given a diagram image, the problem of automatic interpretation involves three key operations [128]:

- Recognizing the visible elements (graphical objects) of the diagram image;
- Detecting the spatial relationship between those elements; and
- Interpreting the symbol-relations arrangement according to the specific syntax of that visual language (diagram notation).

For high-level analysis of diagrammatic communication, the parts of the visual representation which must be known before interpretation can occur, are listed and categorized in Table 3.1.

Referring to Table 3.1, the elements of diagram syntax can be grouped according to those concerned with local elements (the first two categories), and those affecting the global arrangement of elements (the last category). The totality of symbols, their attributes, and their interrelationships define the meaning of the diagram. These three sets of information can be referred to as the concrete syntax of the diagram. In this chapter, we show a way of analysing diagrams to extract the first two syntactic categories required for diagram interpretation.

Category	Elements involved
Symbol–related elements	Symbol types Symbol attributes Symbol position Composite symbols Other visual elements used in the diagram
Relation–related elements	Relations Relation type Relation operands (what symbols use each type of spatial relation)
Symbol arrangement system	Symbol-to-symbol arrangement pattern Special symbols and their placement (ordering), for example start symbol in FSA Diagram reading order

Table 3.1: Syntactic elements of node-link diagrams.

3.1 Extracting concrete syntax from a printed diagram image

Consider the FSA diagram image in Figure 3.1. The image has a raster representation which stores greyscale values of each pixel of the image. This raw raster representation offers little value for a direct high-level interpretation of the diagram. To automatically obtain semantics from such a diagram image, all elements of the diagram and the relationships between these elements first have to be derived from the pixels making up the image.

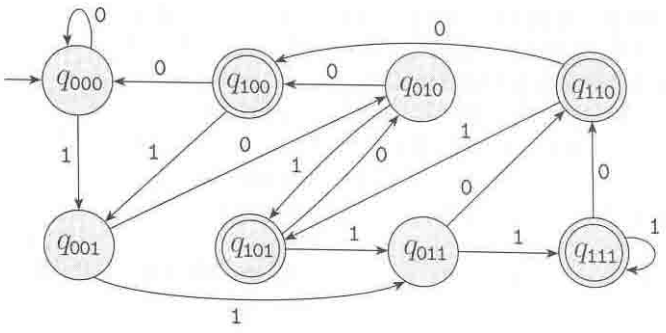


Figure 3.1: Digitized FSA diagram.

Obtaining the concrete syntax from the digitized equivalent of a diagram requires a low-level processing phase to separate objects into layers, identify the primitive objects detected, and return a list of elements and their positions in the image.

This list is required in order to obtain a spatial organization of the elements of the diagram represented in the image. The process of retrieving the underlying syntactic structure of a visual communication object from its spatial arrangement is referred to as spatial parsing [19, 92].

3.1.1 Spatial parsing using the document reverse production process

The most commonly adopted approach to graphical document analysis is to attempt the extraction of information from the 2D representation by reversing the document production steps [94]. The diagram production process starts from image conception, to encoding the conceived idea in a 'formal' representation notation, and then transferring it from a mental concept to a concrete structure to yield a document. The reverse process therefore starts from a document image, analysing it based on its structure and notation to obtain the initial conception of the diagram's author.

In order to achieve the interpretation objective, diagram recognition systems employ a sequence of processes. Some common phases in graphics recognition systems are [14]:

- Early processing, such as noise reduction or de-skewing;
- Segmentation, to isolate symbols;
- Symbol recognition;
- Identification of spatial relationships among symbols;
- Identification of logical relationships among symbols; and
- Construction of meaning.

Our recognition system follows this pattern, having a pre-processing phase, a segmentation phase, a symbol recognition phase, and an interpretation phase. In this chapter we discuss the extraction of symbols, interconnecting lines, and spatial relations; input data which will subsequently drive the higher level recognition phase.

3.1.2 A recognition system motivated by multi-dialect recognition

A diagram recognition system is usually bound to a specific diagram type and would not be useful for recognizing any other type of diagram without some modifications [14, 17].

The focus of our system is FSA diagrams, which are an instance from the node-link diagram class. Although we are concerned with the FSA dialect of graphs, specific knowledge of FSA diagrams are only applied at the higher levels of the interpretation system. For instance, the knowledge that a smaller graphic element located inside a circle is likely a text element is not exploited in the text-graphics segmentation process.

The visual elements used in the production of an FSA diagram are: nodes, which are usually represented by circles or in some cases ellipse; lines, which could be straight, curved, dashed, or polylines; and text. Non-mark elements such as spatial relations are also used in this diagram class.

The set of visual and non-visual elements used in the creation of FSA diagrams are no different from those used in the creation of other node-link diagrams. In fact, many non-standard notations such as textbook illustrations which model hierarchies and networks also use the same set of diagram elements.

Based on the observation that node-link diagrams share the same structural elements, our motivation is to consider a means by which our recognition efforts will not be tied to FSA diagrams alone. The idea is therefore to approach the recognition in such a way that other dialects of graphs or node-link structures may be recognizable.

The result is a diagram recognition system which has two main parts, namely a structural interpretation stage and a semantic interpretation stage. The first four of the phases listed in Section 3.1.1 (see page 35) fall within the structural interpretation stage. The purpose is to apply non-domain specific strategies to obtain the parts and spatial relations of a diagram image. When these elements are obtained, the semantics of the diagram can then be recreated according to the syntax of the specific visual notation by a high-level interpretation stage.

3.1.3 What is in a diagram to understand?

An automated system to interpret a node-link diagram needs to extract the following from the diagram image:

1. Nodes;
2. Node locations;
3. The specific symbol type of each node symbol (the geometric shape used to depict the node). Specifically, the symbols commonly used in node–link diagrams are circles and polygons;
4. Lines;
5. Line origin and terminating points;

6. Directed lines;
7. Directed lines source and target points;
8. Line type (dashed lines, directed line, forked line);
9. Text strings and their locations;
10. Spatial relations between each symbol and the other symbols around and connected to it – specifically adjacency, and connection relations;
11. Logical relations between symbols or groups of symbols; and
12. Logical and explicit groupings of symbols into composite symbols (or entities).

We consider Elements 1–9 above as part of the structural interpretation, which can also be regarded as the lexical analysis of the diagram. The remainder of the processes are undertaken in the domain specific interpretation phase of our recognition system and are covered in Chapters 4 and 5.

3.1.4 Recognition of diagram elements

The structural recognition of a diagram image involves extracting the visual symbols of the diagram, identifying them and detecting the relationship between symbols. The extraction and recognition require that the diagram be segmented into its constituent parts. Each of these tasks (segmentation, extraction, and recognition) has multiple processes involved and the remainder of this chapter details those processes.

3.2 Preprocessing and pixel-level processing

Image processing techniques provide the primary interface to handling and manipulating the raster representation of diagrams. Before primitives in the image are detected, the diagram image undergoes processing that adjusts the image quality for analysis purposes. Improving the quality of the image enhances the recognition results. The algorithms used in image processing and analysis often require parameters to control the operations, and these parameters are dependent on the nature of the particular image.

The diagram in Figure 3.1 is a typical scanned diagram with attendant noise, and other flaws which may be due to poor quality of the original hard copy, low resolution digitization, or both. This diagram instance is an example input image for the pre-processing phase.

Recognition starts with the pre-processing of the digitized image. The pre-processing stage handles the conversion of a diagram image to binary form (binarization), and the removal of noise (noise reduction). In some cases the repair of the image to make it more suitable for recognition (enhancement) may be carried out at this stage. The pixel-level processing stage may also include a segmentation process [121].

In this section we cover our specific choice of techniques and the motivations for those choices, and their implementations. Some of these choices may be biased towards diagram images of average quality. Coverage of advanced document pre-processing can be found in [70, 121].

3.2.1 Noise reduction

In reality, noise will exist in degrees ranging from negligible to severe, and a variety of methods for handling diagrams with severe noise is discussed in [53]. The output of the pre-processing phase is a noise-reduced version of the original digital image. We do not implement a special noise-reduction routine in our pre-processing phase, as our goal is to build a system that processes images from textbooks which are obtained via high-quality scanners.

The example image in Figure 3.2, obtained after a thresholding process has been carried out on Figure 3.1, shows that basic thresholding operations removed most of the random markings and noise which existed in the original image. The main reason is that the values of the noise pixels fall below the selected threshold value.

The diagram (Figure 3.2) and the other node-link diagrams tested in the course of this research did not require substantial noise-handling efforts.

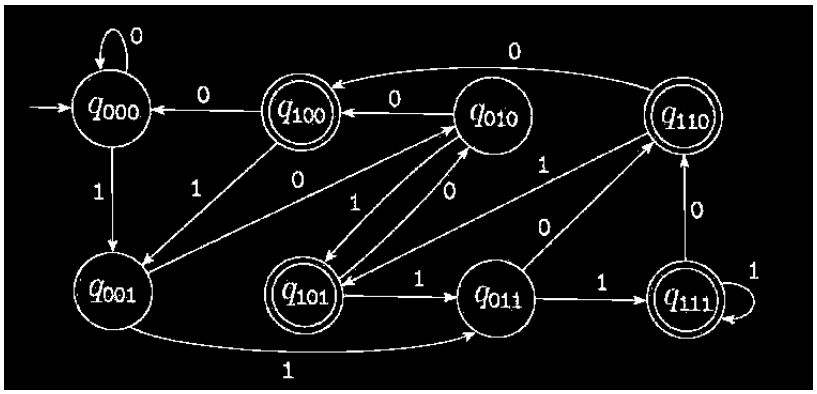
3.2.2 Thresholding

Converting the greyscale-mode diagram image to single-bit foreground or background values is achieved by thresholding the diagram image. The thresholding operation can either be local or global.

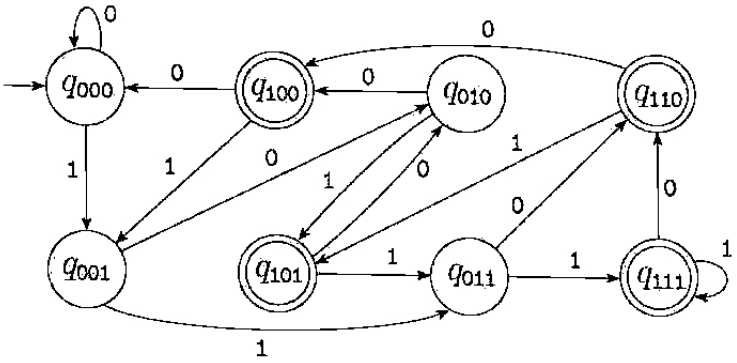
In deciding what the threshold value will be, local thresholding considers the greyscale values of surrounding pixel locations, while global thresholding uses a single threshold value for the entire image. Since most node-link diagrams are produced with strokes having uniform tones which are well distinguished from the white background, a global thresholding method was sufficient for our purposes.

The global thresholding process examines each pixel in the image, compares it with a predetermined threshold value, and then converts the pixel value to either a background (0) or foreground pixel (255) depending on whether it meets the threshold or not.

The selection of the threshold value is crucial, as it will determine the remaining visual information on the image after the thresholding operation. We utilize a



(a) FSA diagram after thresholding.



(b) Inverted image of the FSA diagram after thresholding.

Figure 3.2: FSA diagram after thresholding.

global threshold using a fixed pixel value of 160. This value represents about 40% tone of black. Any pixel which does not have up to 40% black (foreground) is converted to white pixel (background).

A greyscale value of 160 is 40% black foreground and is a safe value to use in thresholding the images given the quality of output in current scanner hardware, and the fact that the diagrams are scanned from text of black prints on white background as occurs in most books. The optimal value to use can also be automatically derived by examining the histogram of grey values in the image; however, the global value proved adequate in all our experiments.

Images produced on paper usually follow the tradition of black foreground on white background, but binary image processing operations often take the foreground as white (pixel value of 255, or *True* for binary image). Therefore, after thresholding, the scanned image is transformed into an inverse with the foreground as white pixels (ON) and the background as black pixels (OFF). Further discussions in this thesis therefore will reflect white foreground on black background.

A limitation of this thresholding stage is that we do not consider diagrams

which utilize shadings as a visual communication element. Due to this limitation, the recognition of diagram notations utilizing shading as a visual element is impacted. There are a number of advanced thresholding techniques in document image processing and these are covered in [70].

3.2.3 Diagram image thinning

Thinning creates an image that makes the analysis of critical features such as end points, junction points, and connections more precise [93]. For obtaining the skeleton image, the medial axis transform approach or mathematical morphological approach can be used. The result of thinning the FSA diagram of Figure 3.2a, using basic morphological operations, is shown in Figure 3.3.

The skeletonization function in the scikit-image package [149] which is based on the algorithm by Zhang [161], was found to provide a skeleton image with better continuity (see Figure 3.4) than the morphology-based thinning operation we considered earlier. The disconnections experienced around junction points in the morphology based thinning operation is not present in the result from scikit's skeletonization function. This issue is discussed in more detail in Chapter 6, where we discuss the results of our experiments.

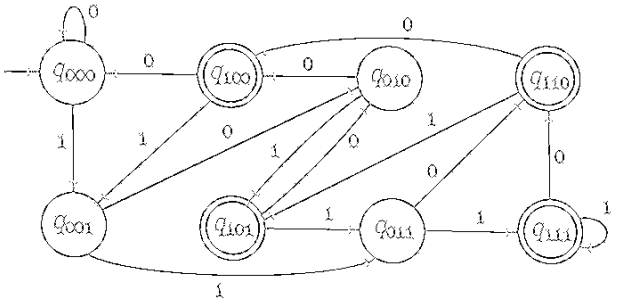


Figure 3.3: Example FSA diagram after thinning with morphological operations.

The behaviour of the thinning operation on a symbol is influenced by the structure of the symbol; that is, whether it is a filled shape or not. Filled node symbols lose their geometric form (for instance a filled circle will be represented by a tiny dot), making it impossible to analyse them by the geometric features used in subsequent stages. This is clearly undesirable. However, in conventional FSA diagrams, graphs with solid nodes almost never occur, and the limitation of this issue will therefore be negligible.

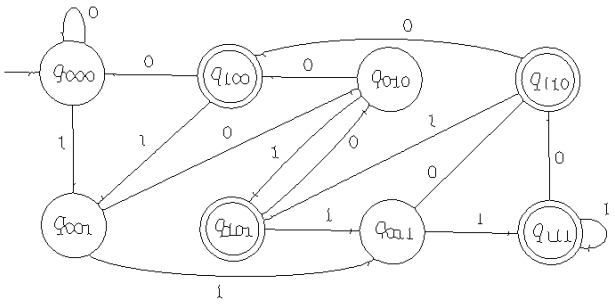


Figure 3.4: Example FSA diagram after scikit-image skeletonization function is applied.

3.3 Segmentation

The input into the segmentation phase of the recognition is a thinned binarized image of the diagram. The segmentation process separates the diagram into text and multiple graphic layers. Segmentation occurs on two levels. First, text elements are identified and separated, leaving graphic elements. Subsequently, on the second level, the graphic elements are separated into node and line elements.

3.3.1 Text-graphics separation

In considering a text-graphic separation technique, the purpose and nature of text occurrences in node-link diagrams need to be examined. In node-link diagrams, text is used as labels, and these labels are often a single character or a few characters at most. This key difference, in contrast to predominantly text documents, causes most text-segmentation algorithms to fail when used on a single character or sparse character labels.

In node-link diagrams text characters may touch graphics elements. Another consideration is that text strings may be orientated at various angles in the diagram; this is a case which seems to occur more often with edge labels in the diagram. The issue at this stage is how to correctly identify and separate all textual elements, considering the factors pointed out above.

There are ample methods for the text-graphics separation process in literature [146], and most approaches utilize the nature and structure of particular diagram notations in order to tackle the text segmentation process. They also utilize geometric features of text like size, stroke density, groupings of character into strings, and the length of linear components of the text string. The range of values of these features for text is quite different to that of graphical elements.

Some additional features specific to text in node-link diagrams are given below:

1. Position:

- Text can be located inside a symbol, which is usually a geometric shape;
- Text can be positioned above, below, to the left or to the right of a directed line in the diagram; and
- Character and text strings can be multi-oriented. The orientation could be vertical or horizontal at any angle.

2. Geometric features:

- The text is typically a single character or a string of few characters;
- Subscript characters could be involved; and
- Text elements are likely to be the smallest connected component present in a node-linked diagram.

3.3.1.1 Classical text-graphics separation processes

Tombre [146] highlighted the steps in the popular text-graphics separation method of Fletcher et al. [58] and enhanced it to cater for text touching graphics. Their text-graphics separation method utilizes connected component sizes, and the density of foreground pixels in the connected component, as measures to distinguish text. An additional shape filtering process selects small elongated items which, due to their size, can be categorized as text, but which may have a chance of being graphics elements too. Later stages finalize the classifications of these elements.

Like any other image object, the connected components of text characters have properties which can be measured. Properties like area, stroke density, horizontal and vertical profile of the connected component are useful to detect whether a connected component is a text character or a graphic character.

The major challenge is the occurrence of a text-graphics merge (see Figure 3.5), and an approach to handle such a situation is given in [146].

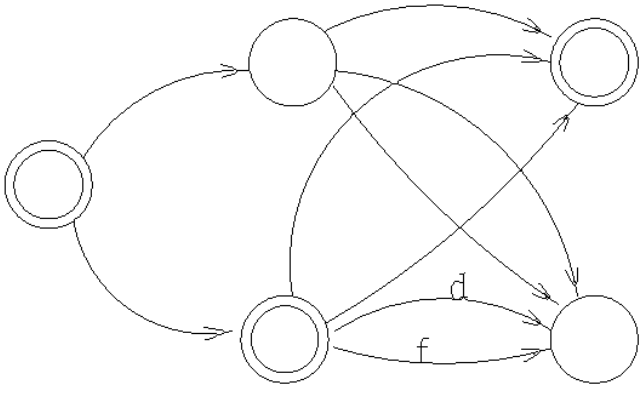


Figure 3.5: Diagram with text touching lines.

3.3.2 Text-graphics separation using geometric features and structural form

In typical FSA diagrams, the labels are strings with one to three characters, and they are often not on the same horizontal level, as a result of the common use of subscripts. Therefore, text-graphics separation processes involving the grouping and collinearity of text characters is unsuited for FSA diagram labels. However, other characteristics of text could be exploited.

Since it is known that the elements in a node-link diagram are text, lines, and geometric shapes, the text-graphics separation problem can be viewed as assigning connected components of characters present in the image to a text class. This is achieved in our case by using geometric and structural features of a connected component such as its size, inner pixels pattern, and the ratio of its area to minimum axis length. These basic properties satisfactorily distinguish text objects from the other objects.

The size of text characters in node-link diagrams is naturally smaller than that of graphics in the diagram. However, putting that knowledge into a (computational) procedure that would assist in the analysis is an issue. This is because text sizes vary. One way to obtaining a threshold size is to take the mean of connected component sizes and use this to identify relatively smaller elements in the diagram. However, the presence of outlier(s) (for instance a massive connected component of conjoined elements) may skew the mean by yielding a result unfit for the analysis. It is therefore important that connected component sizes be checked and such over-sized components not used in the computation.

Another option is to use the median value of the areas of the set of elements in the diagram, since the median value is not affected by the extent of a single value. However, using the median value also does not guarantee a perfect threshold. The default choice in our system is to compare both values, choosing the most suitable value to use as threshold. If the mean value of the connected component sizes (A_m) is lower than double the median value of the same set ($A_n \times 2$), then the mean value A_m is used, else the median value A_n is used as the threshold. The size threshold hence takes into account the possibility of the occurrence of big-sized text characters.

The value of the ratio of the convex hull area of the object C_c, to that of the connected component area, C_a, is another examination to which the connected component is subjected. Just as the connected component area sizes are lowest for text characters, this ratio is also lowest for text components in the diagram.

The advantage of the C_c/C_a ratio is that the size variation of ratio between text and graphics elements is remarkable. Except for relatively large text components which are an exception, this ratio is below 10 for normal sized (average font size of 12 points), for the set of text tested. In the case of graphics, lines have the C_c/C_a ratio at about 15 to 20, depending on the area of the connected component of the

line. For circles, the ratio goes higher, reaching up to 50 in some cases. This ratio, along with other unique features of text components, makes it possible to identify which connected component is text.

3.3.3 Diagram decomposition – segmenting nodes in node-link diagrams

A printed node-link diagram will likely present itself as a massive, single, connected component after text regions have been removed, with the connecting lines and nodes joined together as Figure 3.6 shows. Further processing requires that this graphics be separated into atomic graphics. To proceed with the analysis, the image is subsequently separated into symbols and lines.

After decomposition, an analysis of the features of the various connected components can then proceed to identify which symbol is represented by each graphic element. A description of these processes follows.

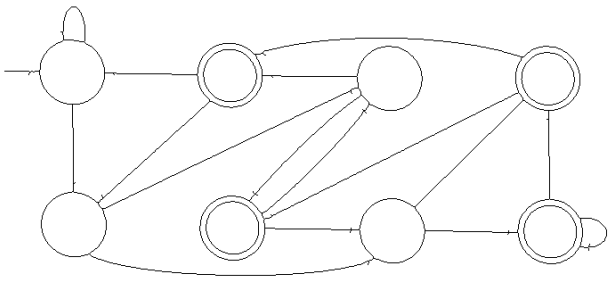


Figure 3.6: Graphics layer for sample FSA diagram.

The original graphic, after thinning and text-graphic separation, contains only connecting lines (straight lines and arcs) and geometric shapes (symbols). The thickness of the lines and symbol boundaries are single pixel. The objective at this point is to have an image in which the shapes (symbols) are disconnected from the connecting lines, and with line arrows and tails no longer touching the respective source and target node symbols. Separate connected components are thereby obtained for atomic diagram units, preparatory to further analyses to recognize them.

For diagrams with filled node types, separation could be done by eroding away connection lines [6]; the operation is uncomplicated and involves using a morphological operator to erode away connection lines. In this way the complicated phase of decomposition of the diagram is bypassed.

The decomposition of the image into its graphic units is necessary for symbol recognition of the diagram to occur. Two processes are involved in the decomposi-

tion stage. In the first step, the junction point is identified and in the second step the line is disconnected from the adjoining symbol.

3.3.3.1 Locating junction points in the diagram image

Since the diagram symbols in node-link diagrams are basic geometric shapes having empty inner regions, it can be safely assumed that the junction points in the diagram are external to the symbols, or lie at the borders of symbols. Therefore, junctions can be used as triggers to identify shared lines and symbol regions.

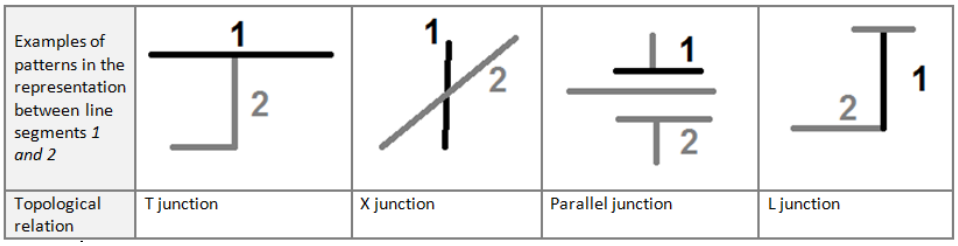


Figure 3.7: Line junction patterns, adapted from [126].

Some common patterns formed when lines intersect with other lines or when lines connect with the boundary of shapes, are shown in Figure 3.7. T-junction patterns are identified and used to drive the decomposition process. The scaled segment of the FSA diagram in Figure 3.8 shows the topology of the T-junctions.

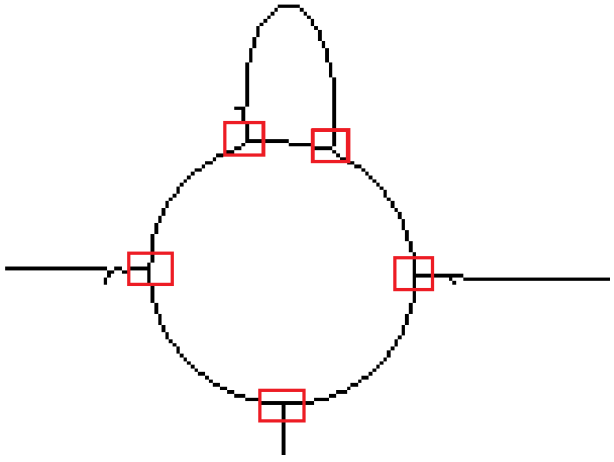


Figure 3.8: A scaled diagram segment showing the patterns at line–symbol junction.

Some properties of the connection points, that aid in correctly identifying them, are:

- There are only a total of eight pixels surrounding a given pixel in the standard raster representation, and therefore the connection line is naturally constrained to attach to a pixel of a symbol boundary via one of the pixel neighbours adjacent to the line path, as highlighted in the areas with red borders in Figure 3.8. The possible connection patterns are used to hypothesize a junction point.
- Using these patterns, false junction points may also be detected as candidates, for instance at line crossings and the corners of polylines (X-type junctions and some L-type junctions).
- There is a minimum of 3 ON (foreground) pixels in the neighbourhood of the junction point being searched for in the raster diagram.
- One or more lines may originate from, or terminate at, the symbol boundary.

Detecting a junction is based on first searching the 3×3 neighbourhood of every ON (foreground) pixel in an image and summing other ON pixels in that 8-neighbourhood. In the thinned diagram, a foreground pixel would normally be surrounded by two other foreground pixels in its neighbourhood. Figure 3.9 shows a rough semblance of this situation. The exception to this is when a pixel is at a junction point, an intersection, or a joint. Figures 3.10 and 3.11 highlight pixels with more than two foreground neighbours, in yellow.

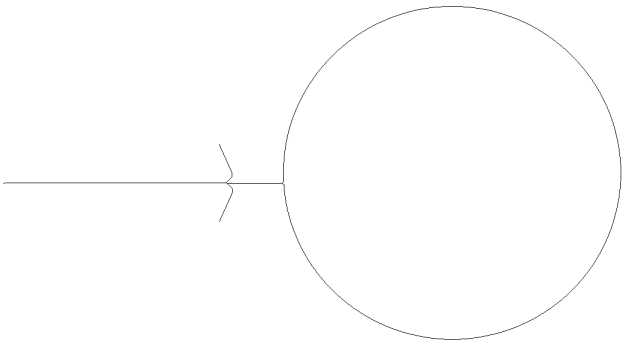


Figure 3.9: Pixel structure for a thinned diagram section.

3.3.3.2 T-junction severance in node-link diagrams

Junctions are formed when nodes and lines unite, with pixels of one joining with those of the other, thereby resulting in a single connected component. Various approaches have been used in literature in order to retrieve the nodes in a diagram when they are conjoined with lines.

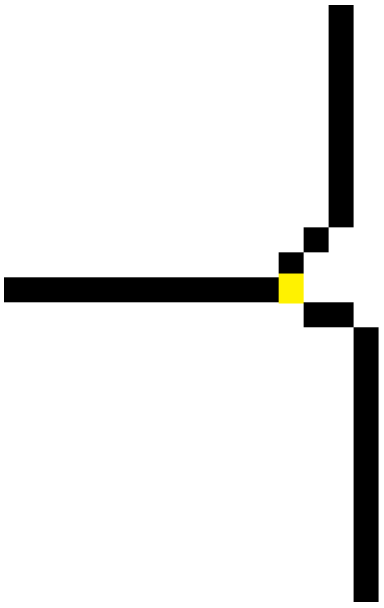


Figure 3.10: A joint pixel in a diagram, highlighted in yellow.

A morphology-based method which works by applying an erosion operation on the diagram to remove connecting lines, was used in the recognition system described in [6]. The process was premised on the nodes been filled shapes rather than shape outlines, therefore diagrams with unfilled node symbols will have the nodes removed along with the connecting lines if this method is used.

The flowchart recognition system [132] considers connected components of background regions of looped objects as potential nodes. The inner regions are separate and easily detected, but since such regions are not unique to node symbols alone, further analysis is carried out to identify which of the candidates is a node.

Disconnecting junctions seem a natural approach to take for node-link segmentation. This requires that junction points in the diagram are first detected, and then at least one pixel removed to break the conjoined connected components into separate units. Locating junction points is straightforward and was discussed in the previous section.

Severing the joint, however, is more complex. The continuity of node borders is crucial for subsequent analysis. If the border of a node is broken at more than one point, then it is no longer a single connected component, and as such the recognition phase would not identify it as a circle, for instance, but separate units of curved lines.

In order to minimize the disturbance of shape borders, the severance process targets pixels belonging to the connecting line instead of the node border. This requires a search around the junction region and an examination of pixel patterns to determine which pixels belong to a connecting line.

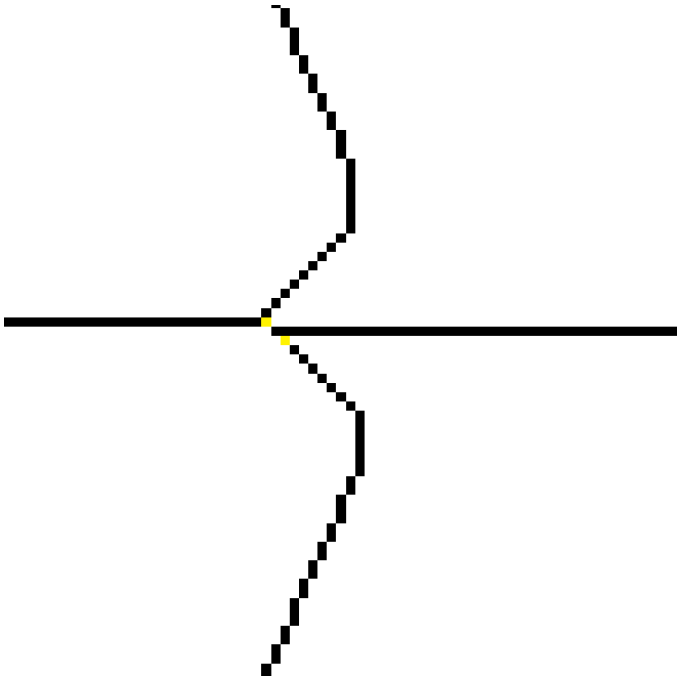


Figure 3.11: Intersection area in thinned diagram section.

The major challenge in determining the correct pixel to severe is that the skeletonization process results in an image with distorted patterns at the junction. This makes it difficult to have a fixed set of patterns which covers all the possibilities that could occur at a junction in the diagram.

3.3.3.3 Detecting connecting line pixels

The identification process identifying a joint pixel does not inform whether the pixel belongs to a node structure or a line structure. This information is necessary to avoid destroying the continuity of the borders of a node structure. Figure 3.12 exposes this problem. In fact, the joint pixel is shared by the line and the node.

Discriminating the set of pixels making up a line from that of a node border is difficult, especially since the decision has to be made based only on a local examination of the junction. The problem of pattern non-uniformity of both lines and node borders in thinned diagrams complicates determining line pixels at the junction region, and also makes creating a set of templates for all possible node-line junction patterns infeasible.

Therefore, to progress, our solution was to assume that the junction patterns are properly formed, at least those of connecting lines. A total of eight *stable line patterns* were consequently defined. These lines are depicted in Figure 3.13. The set of stable line patterns have their roots in the eight pixels of the 8-neighbourhood

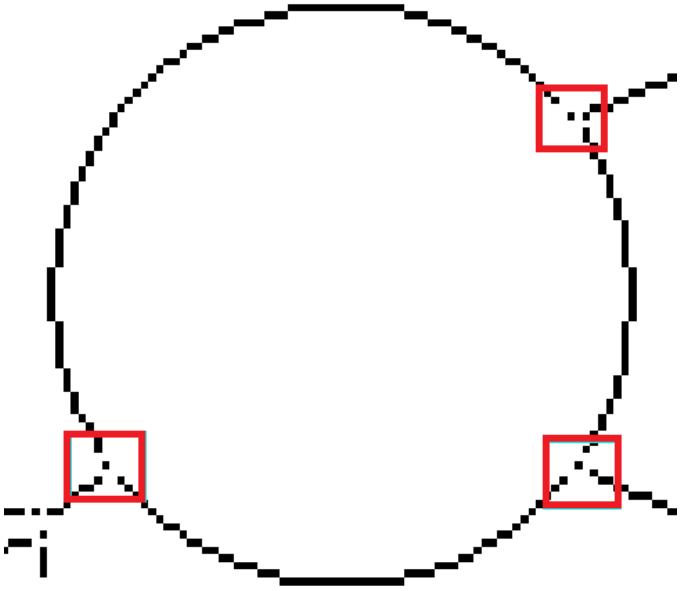


Figure 3.12: Damaged node borders.

of a pixel. These lines are referenced as the 45°, 90°, 135°, 180°, 225°, 270°, 315°, and 360°.

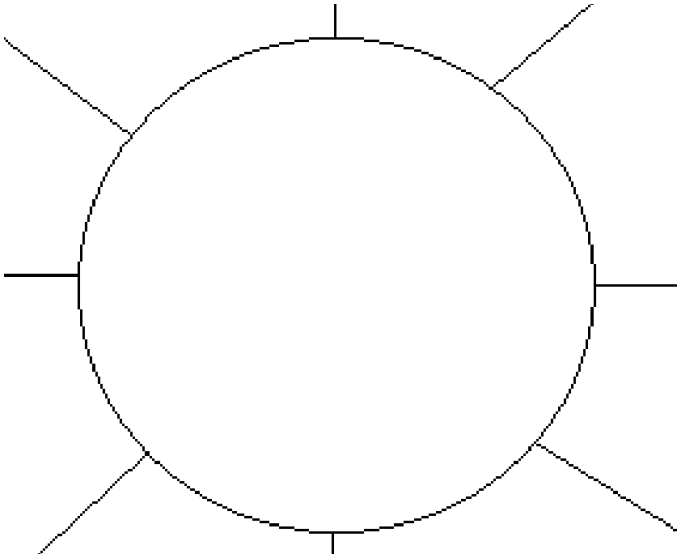


Figure 3.13: Stable connecting line junction patterns.

Each line pattern is divided into three zones which are the anchor, root, and

stem. The anchor is the pixel detected as a potential junction pixel, while the root is the imaginary continuation path of a connecting line assuming it did not terminate at the border of the node. The root lies in the inner regions of a node. The stem is the actual line element. These three zones for the stable 270° line pattern are illustrated in Figure 3.14: the anchor pixel is marked yellow, a segment of the root is marked green, and a segment of the stem is marked red. Our objective is to identify the portions marked red.

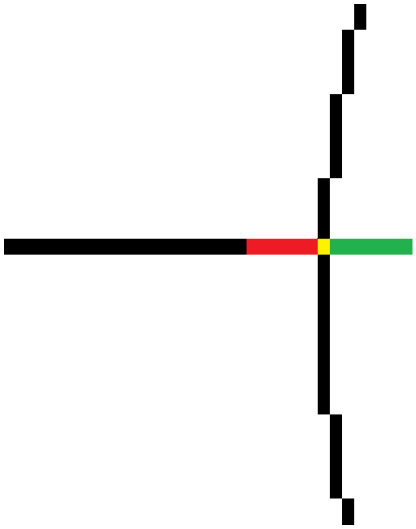


Figure 3.14: Parts of a line pattern (at a 270° line junction).

The pixels forming the stem are present in the image, and the anchor pixel is present, but the roots have to be discovered; discovering the root helps in hypothesizing the pixels of the connecting line. Figures 3.15 and 3.16 illustrate the concept used in assigning labels to the pixels in detected junction regions.

In Figure 3.16, about the anchor pixel position (x, y), pixels $(x, y-1)$ and $(x, y+1)$ are ON (foreground pixel), and both pixels lie in a path of natural direction of a line continuous beyond the anchor pixel.

On the other hand, pixels $(x-2, y)$ and $(x-1, y)$ lying on the stem portion are ON. The natural path of continuity of the line is pixel $(x+1, y)$ which, according to the illustration in Figure 3.15, is OFF (background pixel), thus signaling a background region. The pixel arrangement on the path $(x-2, y)$, $(x-1, y)$ is therefore considered to be portions of the connecting line.

By analysing the pixel arrangement of pixels around the anchor point, the connecting line is found and the terminating pixel (such as pixel $(x-1, y)$) of the line can be deleted. In this way, severance from the node is achieved.

The process described above is applicable to the other stable line patterns, with the only difference being where the root lies, or where the natural continuity is

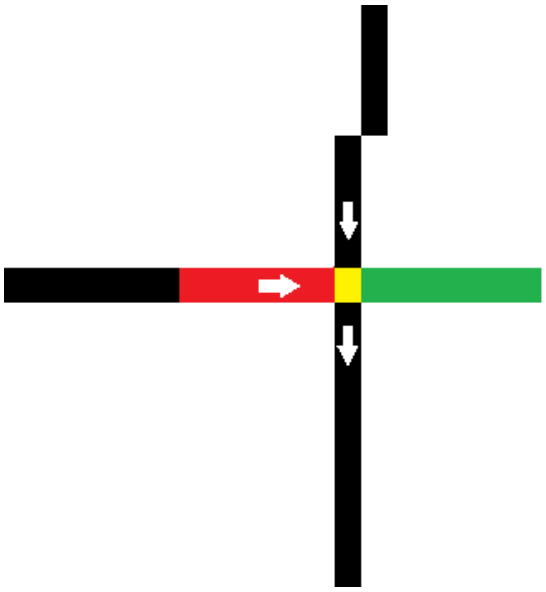


Figure 3.15: Line continuity at 270° line junction.

expected to be for the line. This depends on which of the eight neighbourhood pixels the line is connected through to the anchor pixel.

Since a local analysis is used to detect and separate junctions in the diagram image rather than a global analysis, the detection results are dependent on the local environment; especially the presence of proper patterns at connecting lines and boundaries of symbols.

3.4 Feature-based analysis of diagram elements

After the decomposition of junctions, atomic graphics are obtained from the diagram. Each atomic element of the diagram can be considered as a sub-image of the original diagram. It consists of a particular pixel arrangement, and can be analyzed by examining its structure.

Apart from distinguishing the element by the use of its structure, features of a model object can be measured and used for identifying similar objects that may be in the image. The process by which features are detected and represented is known as feature extraction, and the results of the process are numeric and alphanumeric data rather than pictorial data. Characterizing a shape can be achieved using various schemes; these schemes have been examined in Chapter 2 and a detailed survey exists in [159].

Table 3.2 lists features used in identifying the atomic elements detected in the diagrams. More than one feature is needed to fully characterize an element, as two different atomic objects may share similar values for a feature.

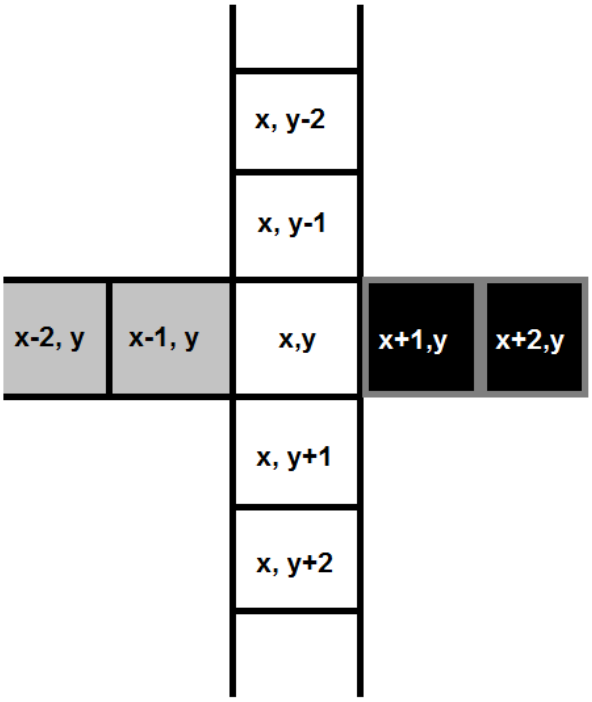


Figure 3.16: Pixel addressing at 270° line junction.

Some objects have known values for certain features. For example, the thinness ratio of a binary object is used as a measure of roundness; *thinness* attains a maximum value of *one* which is the thinness for a perfect circle.

Diagram element	**Features**
Text	Area, solidity, minor axis length
Straight lines	Eccentricity, area, minor axis length, solidity
Curves	Area, minor axis length, equivalent diameter
Circles	Area, minor axis length

Table 3.2: Features used in identifying diagram elements.

In an application, a feature may be used alone, used with another feature to compute a new measure (an example is *solidity*), or used along with one or more other features, thereby forming a feature vector. Our recognition technique is based on combining a number of features. Some new measures which are ratios of two existing measures are also introduced, and these further enhance the identification of elements. The new ratios that are computed, and their value ranges per class, are presented in Table 3.3.

New ratio	Diagram element	Value
Area/M_{axis}	Circle	2.0
Area/M_{axis}	Circular lines	> 1.9 and < 2.0
Area/E_{diameter}	Lines	< 30

Table 3.3: Specific ratios used in identifying diagram elements. M_{axis} is the minor axis length and E_{diameter} is the equivalent diameter of the object.

3.4.1 Feature extraction

The separated diagram elements are identified using the features *area, convex area, eccentricity, extent, convex area to area ratio, area to minor axis length ratio, and area to equivalent diameter ratio*. The identification process obtains these measures for each object in the diagram image, and compares it with models of the elements used in the diagram notation.

In order to maintain similar concepts and definitions with the tool used (that is, scikit-image), the above mentioned properties are considered according to their use in scikit-image. Area is taken to be the number of pixels in the region, convex area is the number of pixels of convex hull image, and extent is the ratio of pixels in the region to pixels in the total bounding box. Eccentricity in this case refers to the eccentricity of the ellipse that has the same second moments as the region. The eccentricity is the ratio of the focal distance (distance between focal points) over the major axis length. The value is in the interval $[0, 1)$. When it is 0, the ellipse becomes a circle [2].

The process checks in order whether the graphic object matches the features of a circle, then that of a straight line, followed by a curved line, and finally text. If it does not match any of the set of recorded features for each of those, it is categorized as a *miscellaneous element*.

The area of the object relative to areas of other detected objects is one of the measures used. The mean of area sizes is a good candidate to compare object size to, but in instances where segmentation has not properly taken place, giant objects may be present. In such instances, the mean size is skewed and if the skew is such that most of the objects have smaller sizes than the mean, then twice the median size is used. Ultimately, double the median size value, or the mean value is used, depending on which of these values is lower.

The eccentricity of a straight line has a maximum value of one, as well as solidity. The ratio *(Area / Minor axis length)* has a value of two for perfect circles, and circular lines have their values lower than two. The value goes up for text and is highest for highly eccentric objects. The values were confirmed in 30 different images containing lines, curved lines, and circles and text.

Diagram element	Features
Circles	Area $>$ (median area size $\times$ 2) (Area / Minor axis length) $\approx$ 2 Eccentricity $<$ 0.5
Straight lines	Area $>$ (median area size $\times$ 2) Eccentricity $>$ 0.96 Solidity $<$ 0.09 (Area / Minor axis length) $>$ 10
Curves	Area $>$ (median area size $\times$ 2) (Area / Minor axis length) between 1.95 and 2.9 Solidity $<$ 0.1, Minor axis length $>$ 100
Text	Area $<$ (median area size $\times$ 2) Solidity $>$ 0.04 Minor axis length $<$ 10

Table 3.4: Decision conditions used in identifying diagram elements.

3.4.2 Symbol recognition in node-link diagrams

The node symbols in node-link diagrams are simple geometric shapes. In particular, FSA diagrams by convention use oval shaped symbols to signify a node. After disconnecting the connecting lines from the nodes, the diagram contains separate connected components of lines and node symbols. It is necessary to detect which connected component is a line and which is a node symbol. To distinguish, we analyse the features of the connected components. Features listed in Table 3.4 are used for discriminating between edges (lines and arcs) and nodes.

After classification into line and node classes, the shape identity of each of the symbols in the node class can be obtained by examining its features and comparing it against the set of features of model shapes known to the system.

Since all nodes in an FSA diagram are depicted using the same shape (circles and ovals), we do not bother to further implement a detailed recognition process for other shapes. For other notations utilizing different geometric shapes, there are varieties of other methods to identify the symbol shape.

Kasturi shows an alternative approach that utilizes loops of minimum redundancy to identify and classify shapes in the image [84]. The closed loops in the diagram image are compared with a library of known shapes. This technique can be used to structurally analyse and identify already separated symbols.

Shape descriptors are also useful for discriminating shapes. A number of descriptors and their nature is discussed in [154] and is a good starting point for selecting suitable shape features to expand the geometric symbol recognition capabilities of the diagram recognition system. This capability will mainly allow recognizing different polygons and shapes.

3.5 Extracting ancillary information from lines and nodes

Lines and nodes are used to encode information mainly by the relationship between a line segment and a node. The *connection* relation is fundamental to this. In finer detail, it is the attributes of directed lines that qualify the relation. The attributes of a directed line include its source (the tail end) and its target (the arrow end). These attributes, along with the corresponding nodes the line segment is connected to, have to be determined.

3.5.1 Arrowhead detection

It is necessary to recognize directed lines in the image as these are fundamental elements of directed graphs. FSA diagrams have arcs and lines with arrows pointing to the target node. The arrows are produced in varying styles and dimensions. Directed lines are frequently depicted by an arrow at one end of the connecting line.

We devised a method to identify the line end with an arrowhead. It involves summing the number of pixels in specific zones of a line. The assumption behind the method is that the end with the higher number of pixels is the end bearing the arrowhead. Since the lines in the diagram are already identified by previous phases in the system, analysing the arrowheads of lines is more direct, and faster than trying to locate arrowheads in the original diagram image. This is because rather than analysing the whole diagram image, individual objects are being examined.

Each line is zoned into four equal segments, and a summation obtained for pixels in the first and last segments of the line. Given the coordinates of the bounding box [Min_{row}, Max_{row}, $\text{Min}_{\text{column}}$, $\text{Max}_{\text{column}}$], a line element is segmented into four zones as follows:

$$\begin{aligned}
&\textbf{Start pixel of } \text{Zone}_1 = \text{Min}_{\text{column}},\\
&\textbf{End pixel of } \text{Zone}_1 = \text{Min}_{\text{column}} + ((\text{Max}_{\text{column}} - \text{Min}_{\text{column}})/4)),\\
&\textbf{Start pixel of } \text{Zone}_4 = \text{Max}_{\text{column}} - ((\text{Max}_{\text{column}} - \text{Min}_{\text{column}})/4)),\\
&\textbf{End pixel of } \text{Zone}_4 = \text{Max}_{\text{column}}
\end{aligned}$$

These two zones are central to the analysis. If the sum of pixels in Zone_1 is greater than those in Zone_4, the arrowhead is hypothesized to be located at the left side of the line and vice versa.

In dividing the line into segments, the horizontal (column) extent is considered, except when the line is strictly vertical in which case the segments are obtained by dividing the rows making up the region into four zones and testing the uppermost and the lowermost segments.

3.5.2 Pairing detected lines with their incident nodes

Every connecting line in a node-link diagram is connected to two nodes, except in the exceptional case of the start symbol. The diagram interpretation process requires the determination of which two nodes are linked together by a particular line. To obtain this data, the production of the particular diagram needs to be considered for cases where lines may either be drawn without touching the node, or drawn touching the node.

This issue is resolved by establishing a search zone around the ends of the line. If the boundaries of a circle fall within the search zone of the line, then it is detected and paired to the line, even if the line does not touch the boundaries.

The search zone is computed based on the length of the line in terms of its row and column spans. To establish the search zone, a *search_range* is evaluated. *Search_range* is given as a pair of values computed according to the row and column values of the line's bounding box. This pair of values, referred to as SR_{row} (the search range based on the rows spanned), and SR_{column} (the search range based on the columns spanned):

$$SR_{row} = (A_{maxr} - A_{minr})/3$$
$$SR_{column} = (A_{maxc} - A_{minc})/3$$

The variables $A_{maxr}, A_{minr}, A_{maxc}, A_{minc}$ represent the extents of the line object; the first two for the row values and the latter two for the column values.

Using a third of the lengths enables one to bypass the use of a fixed threshold value, since it is almost impossible to hypothesize the range of gap space which could exist between a line and the target node. Evaluating the gap based on line lengths offers a solution. However, choosing values relative to line length requires that the allowance obtained not be too large or too small.

If the fact that an arc usually connects two separate nodes on its two opposite ends is extended, the arc can be said to be partitioned into three parts, with the node at each end of the arc 'owning' one third of the arc, and the middle section not been owned by either (belongs to the arc itself). Therefore, a third of the line extent is added to the original extents of the line.

Although this approach seems elementary, using a third of the arc's length as the *search_range* has proven safe, and good results have been obtained in diagrams where the arcs do not touch the nodes.

The two search ranges are added to the bounding box area of a circle to construct the *influence zone* of the line. As a result, each line has an extended area at both ends to search for a corresponding target node.

The search first proceeds by adding the value obtained for SR_{row} to the ends of the line, and then a search for nodes within the new extent is made. The process is repeated but with SR_{column} value added to the line extents.

While there is the potential of duplicate results as a result of 'repeating' this process, it is necessary so as to safeguard against missing some corresponding nodes. One of the *search_range* values may be too small on account of the line having small column or row extents, but the alternative value compensates in such cases.

3.5.3 Analysing spatial relations between diagram elements

A unique feature of visual representations is the liberty of arrangement of graphic elements. By the spatial appearance, position, and arrangement of diagram elements, the graphic communicates with readers. Therefore, to interpret a diagram, the topology of the diagram is analyzed. The purpose of such analysis is to derive all the meaningful relationships that may exist in the diagram as specified by the syntax of the diagram notation.

Visual languages are defined by a grammar and semantics [134]. The visual grammar is composed of the visual alphabet, the visual syntax, and interaction structure. The visual syntax handles the composition of primitives into visual statements. The interpretation of a diagram requires the reading of the primitives of the visual statement according to the visual syntax. Spatial relation analysis is therefore the second key operation in the automatic recognition of diagrams.

3.5.3.1 The challenge of analysing spatial relations

On the surface, the spatial analysis of diagram elements may seem an easy process; since the various elements have been recognized, but that assumption is not true. The complex nature of diagram compositions, which involves multiple types of relationships, and different types of spatial relations, makes the determination of the spatial interpretation of visual sentences difficult.

Selker et al. give four possible categories of composition structures used in visual languages. Their classification focused on artificial visual languages such as those used for the purposes of communicating with computers diagrammatically, and for visual programming. The classification further refines positional elements to eight subcategories. Diagram notations often utilize more than one of these categories to form visual sentences from primitives [134].

The spatial relationships pattern is unique to each diagram notation, making a recourse to the use of domain specific interpretation for the analysis of spatial relations necessary.

3.5.3.2 Fundamental set of binary spatial relations in node-link diagrams

As a means of introducing knowledge required for the computation of spatial analysis, we adopt a strategy of utilizing a subset of diagram spatial relations that seem

	Symbol element	Line element	Text element
Symbol element	CONTAIN	-	CONTAIN
Line element	CONNECTION(TOUCH)	-	-
Text element	-	ADJACENCY	-

Table 3.5: Fundamental spatial relations configuration.

standard to most diagram notations. This set includes the basic relations used in diagram graphics – *contain, adjacency*, and *connection*.

We explain the major spatial relations with reference to FSA diagrams. The *contain* relation covers instances where an object is completely within the extent of another object. The *connection* relation applies to objects in contact via boundaries, and the *adjacency* relation is for objects alongside the left, right, upper, and lower regions of the referent object.

3.5.3.3 Configuration of spatial relations in node-link diagrams

The unique visual elements employed in node-link diagrams belong to three categories: symbols, lines, and labels (in this case, the labels are text, but in some other diagrams such as network maps, the labels could be icons). The fundamental set of relations for interpreting a node-link diagram, and the elements involved, are captured in Table 3.5. Bierman's graphical scanner which was developed to read diagrams, considered this same set of relations [10].

3.5.3.4 Computing spatial relations in diagrams

The bounding box and the centroids of objects are commonly used for computing the spatial relationship between two objects and these two features serve as the basis for the subsequent identification of incident spatial relations in the FSA diagram.

Since diagram symbol objects have been detected and segmented by previous processes, their locations are already known. The problem of spatial analysis is therefore to search for those elements which are in a relation with each other, and for the specific type of relations existing between them. All valid spatial relations between an element and elements of interest in its neighbourhood, according to the fundamental relations in Table 3.5, need to be detected and applied in the interpretation of the diagram image.

The actual evaluation and interpretation of the spatial relations is carried out in the next phase of the interpretation system, which deals with the syntax and semantics of the diagram. It is necessary to opt for the high-level recognition to drive the spatial interpretation, because a rudimentary approach to evaluating, maintaining, and tracking all spatial relationships and relationships between elements in a diagram is difficult.

3.5.4 Concluding notes on structural analysis of node-link diagrams

To achieve the aim of a practical recognition system, the set of primitives and symbols in the diagram image, and their interrelationships, must be obtained. The former is the main topic of this chapter. While we have shown one way in which graphic objects may be recognized, our concluding note is that this is not all there is. Other research exists on each of the stages we have covered, and also on the recognition process as a whole.

A key point being made is a distinction between the phase where graphic objects are recognized and the phase where the diagram is interpreted as a whole. In this way, graphic objects can be extracted from an image using whatever technique is best for the task, and then interpretation can also be carried out with whatever technique is also best suited for that task.

3.5.4.1 Limitations

Image processing and pattern recognition research have mature methods and algorithms to successfully extract the physical elements of node-link diagrams. However, there are instances where diagrams which do not follow basic guidelines of placing diagram elements, are produced by an author. Such diagrams may likely be erroneously interpreted by our recognition system.

In the implementation, we do not recognize dashed lines, crossed lines, and forked lines. There is no lack of algorithms to achieve these [47, 91]. The non-implementation of detection routines on dashed, crossed or forked lines at the low level does not affect the effectiveness of later stages of our system for interpreting diagrams that may contain these. This is because crossed lines are a specific variation of the class of arcs, and we already included arcs and what they represent within the language of graph diagram notation.

3.5.4.2 From pixels to diagram elements

At the end of this major phase of the recognition system, a diagram image will be reduced from an arrangement of pixels to a collection of diagram elements. The new members of the collection include:

- image regions of separated text elements,
- all graphic elements in the diagram (now exists as separated connected components and distinguished by their type),
- a classification of each of the elements (as obtained from symbol recognition, either as node, text, arc, or ‘unknown’), and their location in the diagram,

- information about the source and target node of each connecting line.

These are the basic inputs required for creating a symbolic representation of the original diagram. The collection will be the input into the other major stage of the recognition system, which is the diagram interpretation phase. Using a grammar model, the collection of diagram elements are arranged according to the composition syntax of a diagram notation to form valid visual sentences which can then be further analyzed for an understanding of the diagram structure (semantics).

Chapter 4

Formal languages, visual languages, and the link to diagram recognition

4.1 Introduction

Formal language theory (FLT), "part of the broader mathematical theory of computation, provides a systematic terminology and set of conventions for describing rules and the structures they generate, along with a rich body of discoveries and theorems concerning generative rule systems. Despite its name, FLT is not limited to human language, but is equally applicable to computer programs, music, visual patterns, animal vocalizations, RNA structure and even dance" [57].

Of the three key operations in solving the problem of automatic recognition and interpretation of a diagram image, two operations – the recognition of visible elements and the detection of spatial relations – have been discussed in the last chapter. The third task, which is the interpretation of the symbol–spatial relations organization according to the syntax of a particular diagram notation, forms the remainder of this thesis.

Recognizing and identifying the concrete elements of a diagram does not immediately give the full interpretation of what information is depicted in the diagram. The linear list of primitives and their geometric attributes obtained from recognizing the visual structure of the diagram must be further processed to obtain more information at a higher level of interpretation.

Closely tied to obtaining syntactic entities from the diagram is the acquisition of its semantic entities. While the former is useful for the visual appearance and form, the latter is directed towards reading the diagram's meaning. The visual syntax and the semantics of FSA diagrams are intertwined, as it is the structure of the diagram that gives meaning to the diagram.

For instance, by the syntax of FSA diagrams, an intermediate state is depicted as

circle with a text label inside, and a set of concentric circles with a text label inside depicts an accepting state. States are also semantic elements of FSA diagrams. As a result, the demarcation between the syntax and the semantics of these diagrams seems invisible.

Further processing for interpreting the semantics of the diagram involves an analysis of the syntactic elements of the diagram notation from the extracted primitives. To do this, a specification of the convention of the notation is needed. The specification enables spatial parsing of the diagram from the various tokens in a domain-specific sense.

The specification of visual language syntax is difficult. The difficulty is due to the nature of visual languages where there is freedom of ordering elements in a non-linear manner, the variety of possible relations between symbols which goes beyond the simple one-dimensional adjacency situation of string languages, and the fact that visual representations have directed graphs as their underlying representation and not trees [68].

Like linear languages, the syntactic specification of visual languages is often based on grammars. The advantages of using grammars include the fact that grammars provide a formal definition of the language syntax which allows reasoning about language elements; they are general tools for syntax specification, and their use can be applied to new languages by writing new rules. Since grammar rules provide a structure to the elements of the language, grammars provide a structured mechanism for defining the translation from abstract to concrete syntax, and a grammar specification can form the basis for a parsing algorithm. The latter motivation for grammar use is of particular importance in diagram recognition.

In diagram image recognition, grammars offer a means of formalizing knowledge about a diagram notation in such a way that it can be computed. Since a grammar describes the primitives of a diagram, their attributes, and the acceptable relationship which may hold between them, knowledge about diagram convention is effectively formalized. Although knowledge about diagrammatic notation is only one of the classes of diagram recognition knowledge highlighted in [14], such knowledge is all that is needed to semantically interpret FSA diagrams, and a grammar is valuable to encapsulate that knowledge.

Textual languages have established grammatical approaches for their analysis, with string grammars as an example. The parsing and semantic analysis of languages based on string grammars is standard. In the case of diagrammatic languages, syntax specification is not as straightforward because there are no standard grammars or analysis procedures guaranteed to meet the requirements of every diagram recognition task. The burden is therefore on the researcher to either develop a new grammar system fitting for diagrams, or seek for an existing formalism that fits the structure of the visual language to be described. At the same time, the formalism should suit the nature of the task, which in our case is FSA diagram recognition.

Some diagram recognition research such as [36, 66] implemented their own grammar formalisms; our view is that such will require significant investment of focus and time on the theory of formal grammars, and that is not our primary objective. Secondly, the theoretical basis for the formalism so developed may likely not be rigorously examined, and finally, it may amount to duplication of efforts in a sense. Therefore, examining existing grammar formalisms, proposed by research fields whose main focus is on understanding how to specify and manipulate visual representation, is our choice in this case.

The foundation for the syntactical phase of diagram recognition is laid in this chapter by examining the various treatments of visual languages in the literature, for resolving the problem of graphics recognition and analysis.

4.1.1 Formal languages and visual representations

Diagrammatic representations differ from sentential representations (text languages). While it is easy to derive a model for the description of the grammar of sentential languages by using an operator like concatenation, it is challenging to do so for diagrams. Context-free string grammars, for instance, were not designed to model multi-dimensional (visual) communication and as a result such one-dimensional grammars in their natural format cannot sufficiently describe the symbol-spatial relationship occurrences in a visual language.

Five decades of research into the analysis of images and other visual representations resulted in many grammatical formalisms. In the attempt to describe visual representations, various special grammars using bespoke operators to represent the spatial interrelationships occurring between primitives were also developed. Hence there are many different grammar types for formalizing visual language syntax; to our knowledge, there is currently no standard means of defining diagrammatic languages. This is one of the problems encountering diagram recognition research [17].

Two trends are observed in the different applications employing syntactic approaches; some grammars were used for modeling representations of a singular object in an image (for example, a chromosome image), and others were used to model a system of interacting objects, such as flow graphs. For diagram recognition, grammars which model inter-object representations are required.

Since the use of grammars in pattern recognition and graphics analysis is not new, a variety of grammars exist for the purpose of syntactical analysis of graphic objects and symbols; some examples of these methods are found in [61, 74, 113, 118]. While these grammars are interesting for simple or sometimes complex patterns and shapes, they are not suited for describing diagrams and diagram notations.

We consider images and diagrams as 2D representations, and mention some of the existing grammar formalisms proposed for them. It is impossible to review even a modest fraction of these grammars in a few paragraphs, and therefore we

point out [26, 108] as good resources on the subject of visual languages, and [62] for syntactic pattern recognition in general.

4.2 Grammatical specification of visual languages

Syntactic pattern recognition, graphics recognition and image processing are research areas with associated grammatical applications that would be of interest to a diagram recognition task. However, for applications targeting the interpretation of complete diagrammatic representations, the visual language field has more methods to consider. This may be due to the objective of visual language research, which is primarily concerned with understanding how best to *naturally classify* and *naturally and concisely specify visual languages* [109].

Because the aims and applications of visual languages are broad, there is an abundance of formalisms. This discussion is limited to fairly generic approaches, since our intention is a formalism which can be used within our own recognition system. Visual language theory research offers various grammar models which can be applied to visual programming systems and a variety of applications dealing with visual representations; in-depth reviews of such grammars are found in [69, 106].

Attributed Multiset Grammars (AMG) [68], Constraint Multiset Grammars (CMG) [105], Graph Grammars, Relation Grammars (RG) [41], Symbol-Relation Grammars (SR grammars) [55], Picture Layout Grammars [69], and Extended Positional Grammars [35], are some of the grammars which could specify connection, and geometric relation-based diagrams. Grammars which are modified string grammars (such as the Picture Description Language) [136] are left out, since they are extremely limited in their design and using them in specifying the syntax and parsing a complete diagram is almost impossible.

The survey of visual language specification and recognition in [109] provides a good overview of the different approaches to visual language specification, and it also includes a taxonomy of the approaches with an explanation of the nature of the most popular and interesting ones. By their taxonomy, grammatical specification approaches are classified into generalized string grammars, graph-grammar based methods, attributed multiset grammars, and others not fitting any of the prelisted categories. Several formalisms exist in each of the classes, and the major features and limitations of some of these formalisms are subsequently discussed in this section. It is not meant as an exhaustive analysis of what the grammars are, but rather more of the motivations for our choice of method for specifying FSA diagram syntax.

The key grammar families for visual language specification are graph grammars and attributed multiset grammars [109]. The family of generalized string grammars such as PDL and Positional Grammars is lacking in the aspect of the spatial relationships they are able to represent.

Extended Position Grammars (xPGs), another formalism from the generalized string languages family, extend Positional Grammars (PG), a context-free grammar based formalism. xPGs have string-like productions, with the productions inducing a scanning order of the symbols during the parsing process. This approach yields an efficient parser of the input visual sentence. The parser is based on the LR parsing technique [34]. However, xPGs are also restrictive in the type of structure they can successfully describe. Furthermore, writing production rules in xPGs is complicated. And as much as possible, a grammar which supports writing rules in a relatively natural way is advantageous.

The major disadvantage of generalized string grammars remain, namely, that they have limited expressiveness which is not sufficient for FSA diagram interpretation. Graph grammars and attributed multiset grammars provide better means of specifying visual languages.

Graph grammars are a popular formal grammar across many research disciplines, and different graph grammars have been proposed and studied. The consensus is that graph grammars are powerful formalisms for the specification of visual languages, but their efficiency in terms of computational costs is poor [19]. Some challenges in applying graph grammars are mentioned in [52].

A well known problem with graph grammars is the computational cost attendant with parsing them. However, there have been several attempts to work around or limit this disadvantage; an example is the Layered Graph Grammar [129]. The need for some context-sensitivity while specifying visual languages is also a limitation of graph grammars as most grammars in this family are context-free; this issue has been addressed to a certain extent [160]. Therefore, getting a usable graph-grammar based system in our case, is not totally impossible, but is challenging.

Graph grammars have been used for diagram recognition purposes, with a majority of applications utilizing graph transformation in the mathematics and music notation domains; as observed by [15]. The opinion in [15] is that graphs are better than multisets for diagram processing, because diagram recognition requires complex computations to find the important spatial relationships, and an explicit representation of those relationships makes accomplishing the task easier.

However, some researchers are of the opinion that the initial processing required to obtain the relationships occurring between symbols as required for using a graph grammar is a disadvantage, as an initial graph has to be constructed for the representation, creating an extra task in the system. Also, large grammars become intellectually unmanageable, and noisy and uncertain data (for instance erroneous recognition) are currently not handled well by graph grammars [52].

Clearly there is a contest between graph grammars and attributed multiset grammars with a keen argument for each of these two. The intensive debate is not only between researchers in the two camps, but also among users requiring a grammar for use in their various applications. On the side of users, graph grammars are uninviting given the learning curve challenges, coupled with the confusion attendant

with its use [16].

Considering attributed multiset grammars (AMGs), generally speaking they cannot be recognized efficiently as they are too expressive [68]. Recognizing them therefore requires the introduction of various restrictions. For instance, Picture Layout Grammars [68] which are a restricted form of the AMG class, can be efficiently parsed. Efficient parsing is obtained in PLGs by bounding the domain of attributes that are used in the production rules. Attributes are used in PLGs to represent the spatial layout of symbols occurring in the original two-dimensional representation.

Constraint Multiset Grammars (CMG) extend PLGs by allowing the parsing of a larger variety of diagrams, and also provides a form of generative semantics [105]. It is known that a practical parser is obtainable for CMGs and an algorithm for incremental parsing of CMGs is given in [105]. Furthermore, some steps which are under the control of the grammar writer, such as ensuring that the grammar is deterministic, and that productions are cycle-free (that is, no production can rewrite a non-terminal into another non-terminal) leads to an efficient parsing process for CMGs. CMGs have been used in a significant number of applications [29, 30, 31, 79, 80, 102, 103, 112, 137].

Relation Grammars (RG) are also based on multiset rewriting. Symbol-relation grammars are a form of RGs, and view a visual sentence as having a set of symbol-items and relational-items. A set of rules exists for symbols, and a set exists for relation rules over those symbols. The derivation of the sentence is performed by rewriting both symbol-items and relational-items using context-free styled rules [55]. For a class of SR grammars referred to as boundary SR grammars, an efficient parser is also obtainable [55].

In diagram recognition applications, the interest is more on recognition rather than the generation of visual sentences. As a result, the concern about which approach is suitable revolves around considering what atomic elements (primitives) a specification is based on, the predefined set of spatial relations or means by which spatial relations among symbols is represented, how naturally the structures and substructures of the notation can be formulated as production rules, and the efficiency of carrying out the recognition operation (spatial parsing) using the formalism. An appropriate grammar choice must be able to specify the diagram language, and an efficient parsing procedure must be possible for such grammar.

The various formalisms already mentioned are based on atomic elements at the same level of the primitives of FSA diagrams (geometric symbols, lines, and text). The notable differences between formalisms are how these primitives are treated, the relationships between primitives, and the limits or freedom placed on the relationships and the primitives.

If there are limitations with these approaches with regard to diagram recognition, it is due to the fact that most of them were not designed for the purpose of, or with a consideration for, analysing document diagrams. Rather, they are de-

signed for parsing diagrams obtained from controlled environments such as visual programming editors and not arbitrary diagrammatic representations. It is our observation that specifying diagrams using the formalisms previously discussed is possible, but it all depends on the type of diagram, and possibly some modification to the approach. The ideas behind them are no doubt valuable when syntax specification and parsing of diagram representations is required.

Theoretically, a visual language can be described by any of the formalisms earlier discussed in this section, but the question is which of them is the suitable option for our diagram interpretation task. The motivation for choosing an appropriate grammar formalism for modeling a visual notation are influenced by two factors: how easily the grammar formalism describes the language, and the needs of the visual language implementer [33].

In summary, CMG was chosen for the purpose of processing the output from the diagram image analysis (or lexical analysis) because it possesses the necessary core capabilities and supporting features for the manipulation of visual elements and relationships in FSA diagrams, compared to the other formalisms. The expressiveness of the CMG formalism is enough to specify FSA diagrams without the need for workarounds or enhancements. Specifying diagrams in CMG is also straightforward, and sufficient information exists in literature to be able to apply CMGs in a practical application. We return to discussing CMGs in detail in Chapter 5.

Wittenburg's paper on visual language parsing [152] concluded *"the field really must agree on standard formalisms (Constraint Multiset Grammars is one candidate) as well as standard representations for input. The continuing proliferation of ill-understood variations on visual language formalisms is not helpful"*; and that conclusion could justly end this section.

4.2.1 Some existing formal language models in diagram interpretation systems

Syntactic methods, such as the one we propose for FSA diagram interpretation, are one of the frameworks for diagram recognition [14]. The blackboard system, schema-based systems, and graph rewriting are other strategies used in recognition systems. The grammars highlighted in the previous section were not explicitly developed for diagram recognition, and that raises a question of what grammars exist within the graphics recognition research field for the analysis of various diagram notations.

In diagram recognition, formal grammars have been used in different ways. Grammars have been used to control the recognition process, to store diagram notation conventions (using rewrite rules), to recognize parts of the drawing such as the dimensions of an engineering drawing, or to control the entire recognition system [17]. Furthermore, different grammar types may be required for different

parts of the recognition system. For example, one formalism may be used for the visual part of the diagram and another for the logical representation of the diagram [39]. Grammar formalisms and diagram recognition are further surveyed in [17].

There are notable works in the syntactic interpretation of a diagram from its primitive symbols. Two notable diagram recognition systems which made extensive use of grammars, applying their own grammar formalisms to parse diagrams of different notations, are the Extended Position Formalism (EPF) [36] and Graphics Constraint Grammars (GCG) [64, 65, 66]. These grammar models were proposed and applied to the recognition of a number of graphics domains and diagram notations.

GCG was used by Futrelle in his diagram understanding system to describe and analyze diagrams. The primitive symbols the grammar is based on are lines, polygons, and positioned text. The GCG has a collection of rules comprising a production, a set of constraints, and a set of propagators. The constraints consist of spatial relations and type constructs restricting object types. A set of propagators describe the relationship between the attributes of the rule object and the attributes of its constituents. GCG parsing is handled as a constraint satisfaction problem rather than as a classical parsing procedure, and the output of the parser is a frame-based knowledge representation of the content of the diagram. Futrelle showed how the grammar can be used to parse line graphs, finite automata diagrams and other graphics found in biology texts [64, 65].

DMOS (Description and MOdification of Segmentation) is designed to be a generic recognition method for structured documents. It has been demonstrated for the recognition of document tables [38], musical score [36] and tennis court images in video [37]. DMOS uses Enhanced Position Formalism (EPF) grammar developed by the authors in the analysis of graphic images.

EPF can be regarded as a sort of description language for structured documents and enables the graphical, syntactical and semantical description of a class of documents. The EPF formalism is an extension of the one-dimensional Definite Clause Grammar (DCG), and adds additional operators to the single operator (concatenation) used in the original DCG formalism. The graphic primitives manipulated by this grammar are line segments and connected components obtained from low-level image processing steps.

The demonstrations of EPF and DMOS as reported by the respective authors make them attractive options for the purpose of analysis of diagrams. Opting not to use any of these two formalisms is due to a combination of factors. The first is our desire to use a grammar model which can be easily specified and maintained, which can handle fairly high-level primitives such as geometric shapes, lines and text, and possibly model a major portion of the elements of visual languages as discussed in [134].

Secondly, these formalisms (EPF and GCG) have not been widely and rigorously

applied as much as CMGs, and so there are insufficient details to conveniently consider and implement them. Finally, CMG includes special features such as an existential operator, constraints, and an error-correction capability; features which can potentially enable robust handling of diagram recognition tasks. These details are further examined in the next chapter.

Chapter 5

From symbols to diagrams

"Once we have lexically analyzed a program into a stream of tokens, we are now faced with the more challenging task of parsing the token stream into a formal structure. In other words, we have to figure out how to group the tokens into language constructs like variable declarations, statements, expressions, and so on. These grouping and classification tasks can be done by attempting to match the token stream on some predefined set of rules known as a grammar" [119].

The above excerpt, although a discussion on compilers, summarizes what is done with the extracted 'tokens' from Chapter 3; namely text, circle, and line objects, as well as their respective geometric information.

Parsing FSA diagrams requires a formal specification of FSA diagram notation. For the purpose of parsing FSA diagrams, grammars based on the Constraint Multiset Grammar (CMG) formalism are defined. A grammar for a diagram notation specifies the constructs into which the set of tokens are grouped, and how the grouping occurs.

Through parsing, states (normal state, start state, final state) and transitions, which are semantic elements of FSA diagrams, will be obtained from extracted diagram primitives. For a different diagram notation, these same primitives would likely be grouped into different constructs according to the specific syntax and semantics of the particular notation.

This chapter outlines CMGs and the unique features of the grammar model. Subsequently FSA diagrams are examined in terms of morphology, syntax and semantics. Thereafter, a grammar specification for FSA diagrams is presented.

5.1 Constraint Multiset Grammars (CMG)

Constraint Multiset Grammars have been proposed as the basis of a non-linear (visual language) counterpart of the Chomsky hierarchy [107]. CMGs have been

well investigated and compared to other visual language specification methods such as Positional Grammars, Relation Grammars, and Unification Grammars in [106]. CMGs are an extension of Picture Layout Grammars (PLG), but CMGs cover a larger class of visual languages.

CMGs are high-level declarative languages for visual language definition. They have been used in the analysis of different complex tasks; the ability to handle complex representations is partly due to the fact that CMGs rewrite multisets of tokens. These tokens may have a variety of relationships besides an adjacency relationship (sequence), with one another.

Another major feature in CMGs is the dependence of the grammar productions on constraints. Constraints are used to specify the topological, geometric, and semantic relationships between graphic objects, or collection of graphic objects, in a diagram. A constraint solver applies geometric constraints to maintain diagram semantics.

The subject of suitability, specifically how CMGs fit the diagram recognition task is explained in this section.

5.1.1 Formal definition of CMGs

A formal definition of CMG is given by [105]. A Constraint Multiset Grammar over a computation domain D consists of

- a set of terminal symbols T_T,
- a set of non-terminal symbols T_{NT},
- a distinguished start symbol $S_T \in T_{NT}$,
- a set of productions.

Each symbol $T \in T_T \cup T_{NT}$ has a sequence of attributes. The start symbol may only appear on the left hand side of a production.

Productions in CMG have the form [106]:

$$X ::= X_1, \ldots, X_n \text{ where } C.$$

In this definition of a production, it is assumed that $X \in T_{NT}$, and $\{X_1, \ldots, X_n\} \subseteq T_T \cup T_{NT}$. Moreover, rewriting X into $X_1, \ldots, X_n$ is possible if the context $X'_1, \ldots, X'_n$ exists in the sentence, and the attributes of all the context symbols satisfy the constraint C.

A CMG token $T(\vec{\theta})$ is a type symbol and an assignment to the attributes of that symbol [105]. Diagrams can be considered as collections of graphical tokens such as lines, circles, and any other diagram primitives [20].

A *multiset sentence* is a multiset of tokens and a *terminal sentence* is a sentence containing only terminal tokens.

5.2 The significant features of the CMG formalism

The CMG grammar formalism has a number of significant features in the manner in which it represents pattern structure. A CMG grammar instance consists of the symbol definitions and production rules in similarity to other grammar formalisms, but the design for using these grammar components differs significantly in CMGs.

Each of the unique features of CMGs (symbol types and attributes, positive and negative constraints, existential quantification, and 'external' functions), contributes to either the effective specification of the visual language grammar or towards an effective parsing process; as such they are important to a grammar writer. This section considers each of the features.

5.2.1 Alphabet symbols

Like every formal language, a CMG-based grammar defines a set of terminal symbols T_T as well as a set of non-terminal symbols T_{NT}. For example, in FSA diagrams, the terminal type symbols are graphic primitives (text, lines, circles) obtained from the diagram structure recognition phase described in Chapter 3.

One notable difference between symbols in CMGs, compared to other visual grammar formalisms, is the presence of attributes and constraints over these attributes. These constraints (over symbols) define relationships between a diagram and its components (elements) [105]. Table 5.1 shows the different types of symbols for the FSA grammar.

5.2.1.1 Symbols and symbol types

Every symbol in the grammar has a type, and the type has attributes associated with it. Types may be terminal or non-terminal. In the CMG for an FSA diagram, terminal types refer to graphic primitives.

In CMG grammar notation, the statement $S : T$ specifies that a symbol S has type T, while $S.A$ indicates the attribute A of a symbol S. For example, $ocircle : circle$ defines the existence of a symbol with the name *ocircle* as a symbol of type *circle*, while $ocircle.radius$ refers to the attribute, radius, of the circle type. The name of the symbol is an example of the symbol's semantic attribute.

Each symbol type has zero or more attributes, and the attributes usually describe the geometric properties of the symbol. Constraints are specified over type attributes to define relationships between symbols in the diagram.

5.2.1.2 Symbol attributes

Engelhardt's framework for the analysis of syntax and meaning in maps, charts, and diagrams [50] conceptualized graphics representations as being graphic objects. A graphic object could be elementary or composite; if composite, it consists of:

Symbol type	**Attributes**
Directed lines	*startpoint, midpoint, endpoint*
Circle	*midpoint, radius, area*
Text	*midpoint, label*
Labeled arcs	*startpoint, midpoint, endpoint, label*
State	*midpoint, radius, area, name, kind*
Transition	*from, to, label*

Table 5.1: FSA symbol types and their attributes

- a graphic space occupied by it,
- a set of graphic objects, which are contained within that graphic space, and
- a set of graphic relations in which these graphic objects are involved.

These graphic objects have visually perceivable attributes which are referred to as *visual attributes*. Examples of visual attributes are size, shape, orientation, and spatial position. CMGs enable the modeling of these attributes, making it an attractive option for the specification of diagram syntax.

Each type in a CMG grammar has its individual attributes, and symbols of a type have these attributes instantiated. There are two categories of attributes in CMGs, which are the geometric attributes and semantic attributes. Examples of geometric attributes are *radius* of a circle, and *location* of the circle while *name* of a state, and *type* of a state are semantic attributes.

The presence of geometric and semantic attributes extends the possibilities available to a grammar writer utilizing CMGs. This is necessary given the diverseness of how different diagram notations utilize visual elements to encode information.

In production rules, attributes must be well typed. That is, the attributes of variables must match the attributes of the variable types [29]. The attributes of the symbol on the left hand side of a production are obtained by an assignment from its right hand side.

Constraints over attributes are used to steer the applications of the grammar production; if the constraint is satisfied, then the production rule can be applied. Table 5.1 shows some attributes of various terminal and non-terminal types in an FSA diagram.

5.2.2 Constraints

Beside robust attribute definitions, CMGs employ the extensive use of constraints. A constraint describes the composition for which a production is applicable. The arguments of constraints are terms built from variables, functions, and constants from a computation domain. The constraints and functions operating over values

of constraints in CMG must be computable, and as such constraints include linear and non-linear inequalities over the real numbers and over tuples of real numbers. More complex constraints can be built by using logical operators such as *AND* and *NOT*.

By using constraints, the relationship between a diagram and its components are defined, and conditions for the application of production rules established. For a non-terminal to be recognized from a multiset, the attributes of all tokens involved must satisfy the given constraints in that production.

The constraints of CMGs' enable information about spatial layouts and relationships to be naturally encoded in the grammar [105]. Using attributes of types and geometric constraints, the spatial relationships in the diagram are maintained. The full power of CMGs to specify the syntax of diagrammatic notations thus lie in the proper usage of constraints.

Two types of constraints exist: negative and positive constraints.

5.2.2.1 Negative constraints

Negative constraints are used to ensure the non-existence of certain symbols in the collection of symbols being examined for a production rule to be applicable. This enables determinism in the CMG-based grammar, as it prevents the inadvertent application of a potentially wrong production rule.

A rule may fit a collection of objects, but may not be the correct rule to use, since not all eligible rules will always be valid rules. When a non-terminal features a collection of primitives, and another non-terminal includes a similar arrangement of primitives as part of its own collection, then the danger of ambiguity is introduced. Ambiguity is excluded in a grammar by using negative constraints as it further clarifies what it is the production aims to define.

For instance, a state symbol is a non-terminal, made up of a text terminal symbol, located within a circle type. However, an accepting state non-terminal symbol also fits that rule. Negative constraints can be used to eliminate this confusion by adding a condition that a second circle with same midpoint does not exist in the sub-diagram. Such ambiguity, which is one problem in the diagram recognition process, is therefore eliminated. Without negative constraints, it is difficult to specify many naturally occurring visual languages [29].

5.2.3 Existential quantification

Constraints may require the existence or non-existence of particular tokens as condition for the application of a production [29]; thereby involving other sub-diagrams of the image in the production. Such existential constraints allow to check for the presence of certain sub-diagrams in the collection, without them being reduced.

For example, one way of specifying a transition is to describe it in terms of the presence of a directed line, and two incident states. If this is written as a CMG grammar production in the following snippet:

$$TransI : transition ::= LineJ : directedline, StateK : State, StateL : State \text{ where } (\\ ...\},$$

the two states *StateK* and *StateL* will be reduced upon the application of the rule. Obviously this is not the behaviour intended by the grammar writer. An existential quantification is used to avoid a situation similar to this. It checks if a sub-diagram exists before the production is applied, but does not reduce it.

This feature of the grammar introduces context-sensitivity. It has been observed that natural visual languages are context-sensitive: Marriott [106] noted that graph-based visual languages are not context-free and therefore existential quantifications are useful in the specification of such languages.

5.2.4 Spatial relations in CMGs

There are no explicit representations for spatial relations in CMGs, in contrast with other grammar formalisms where only a pre-defined number of spatial relation operators, or meta-symbols representing spatial relations, are used. This is a key advantage of CMGs over a significant number of other grammar types. Rather than having an explicit set of such operators, CMG spatial relations are established by the constraints in the grammar productions.

The objects in a multiset representing a diagram must contain a representation of their geometry in terms of the coordinate information of the object [20]. The constraints encode the spatial layout and relationship between a diagram and its components. CMGs are therefore able to model spatial relations, without pre-defining any particular sets of spatial relations, by using constraints on object attributes.

The spatial relation *containment* existing between a character label and a state in an FSA diagram is captured by the CMG grammar snippet:

$$StateX : state ::= CircleY : circle, TextZ : text \textbf{ where } (\\ \textbf{CircleY.midpoint} == \textbf{TextZ.midpoint} \\ ...\}$$

The constraint statement **where CircleY.midpoint == TextZ.midpoint** establishes the spatial relation between the text and circle elements in this particular production. A similar production can be designed for the case of containment

spatial relations in FSA diagrams where states are oval-shape; this also applies in other productions where we assume states are circles.

5.2.5 Parsing

A parser for a CMG grammar repeatedly reduces objects which match the right-hand side of a production to the object(s) specified by the left-hand side of the production. Objects match if all the constraints of the production are satisfied [73].

Parsing with multisets is harder than parsing with formalisms based on strings. It has been shown that for CMGs which are *cycle free* in the sense that no production can rewrite a non-terminal into another non-terminal, the membership problem is NP-hard [105]. It is known that a practical parser is obtainable for CMGs and an algorithm for incremental parsing of CMGs is given in [105].

Although the majority of parsing applications of CMGs dealt with online diagrams [103], we believe that a CMG parser for offline diagrams, such as those being considered in our work, is less of a challenge than parsers for interactive diagramming. The reasons for this submission are two-fold. The first fact is that in offline diagrams, incremental parsing is not needed as the whole diagram is already produced and therefore the parsing is relatively easier. The second reason, which is linked to the first reason, is that the challenge of continual checking and maintaining of the integrity of the parser state is absent in static parsing, since there are no instances of the user modifying incomplete diagrams.

5.3 Morphology, syntax, and semantics of FSA diagrams

In Chapter 3, the tasks involved in the interpretation and understanding of diagrams were enumerated, and the stages involved in the recognition of diagram elements analysed. In the same chapter, the recognition of the elements of an FSA diagram from a raster image representation was carried out.

However, it is the global arrangement of graphic objects in a diagram that determines the information conveyed by the diagram, and not only the individual primitive elements. For interpreting the global arrangement, a priori knowledge about the diagram notation is required. Not only that, a means of representing the knowledge in a way that could be used for interpretation is necessary. A syntactic option allows capturing the syntax of a notation, and at the same time provides a means of using the captured syntax in an analysis of the pattern.

The syntax of a diagram notation defines the elements used and how elements can be combined. The semantics of a diagram defines the meaning of the individual elements and their valid combinations.

In this section we give a brief, informal discussion of the form, syntax and semantics of the FSA diagram notation and proceed to writing a formal grammar based on CMGs, for grouping individual graphic objects into higher (semantic) constructs.

5.3.1 Elements of FSA diagrams

Diagrams are birthed from mental concepts in the mind of a writer, and a concrete representation of that mental concept is created by representing it in a written and structured visual form.

As long as readers are familiar with the notation used in producing the concept, they are able to read the structured representation and at least grasp a picture of the original concept in the writer's mind as depicted in the diagram. This structured visual form, which is made up of the visual elements, their arrangement, and the encoding method, forms the syntax of that system of communication. We proceed to examine these syntactic elements of FSA diagrams.

5.3.1.1 Morphology of FSA diagrams

The graphic form of an FSA diagram comprises an orderly arrangement of circles, directed lines, and text. Circles are labeled with text, the text typically being positioned in the middle portion of the circle. Directed lines are straight or curved lines (arcs) with an arrow at one end; the end with the arrow is adjudged as the target end while the tail of the line is regarded as the source end. Directed lines are labeled with text which is usually positioned about the center point of the line, to the top, bottom, left, or right regions.

As directed lines, circles, and text are the graphic primitives for FSA diagrams, they constitute the terminal symbol types for the FSA grammar. The respective attributes of these graphic objects are given in Table 5.1.

The graphic primitives are extracted at the earlier stage of the diagram recognition system. Similarly, the geometric attributes of these primitive elements would have been previously extracted from the diagram image, including the coordinates of the bounding boxes, and the centroids. The syntax is specified in terms of these terminals and their attributes.

5.3.1.2 Syntax of FSA diagram notation

FSA diagrams are drawn according to a set of conventions covering the types of graphic objects used to produce them to how these objects are arranged. Like any other formal diagrammatic form, the rules guiding the production of FSA diagrams form part of the diagram language. These rules ensure a standard in the creation of FSA diagrams.

As a result of those conventions, even if an FSA diagram is not explicitly labeled as one, users familiar with FSA diagram syntax would recognize the diagram as the image of an automaton. This uniformity in the structure of diagram instances from the same notation family, which is due to an adherence to syntactic rules and conventions, is an important feature for automatic recognition of diagrammatic forms.

Some conventions used in the production of FSA diagrams are:

- **Rules regarding states**
  States are drawn using closed polygons; by convention, circles are used. Ovals may also be used. Although the geometric size of a state does not contribute to the semantics, the size of the circle is typically kept uniform in the diagram. For easy reference, states are labeled with text characters, with the label positioned approximately about the midpoint of the state.

  There are three kinds of states in an FSA diagram, and the visual depictions of these states are described in Table 5.2. It is assumed that all states have an incoming transition, and no state exists isolated from all others. All states in the diagram are connected to at least one other state by transitions.

- **Rules regarding transitions**
  Transitions in an FSA diagrams are represented as labeled directed arcs (arrows). Directed arcs are labeled with text characters representing the input from one state to another. Transitions exist between states and always have a source and target state; as such, there cannot be a 'floating' end (except for the start state arrow).

- **Spatial relations rules**
  The permitted arrangement of the graphic objects in the FSA diagram depend on the object concerned and the referent object involved. Two diagram objects have relationships based on their individual object type. For instance, the legal relationship between a circle and another circle is *containment*; such a relationship is found in the accepting states. In all, the valid relations for objects are *containment, connection* (touch), and *adjacency* (near).

  The *connection* relation is exhibited in the state-arc composition, with the extreme ends of an arc interconnecting with a point on the state's border. The *adjacency* relationship covers the analysis of proximity between an object and its neighbour(s), such as the line-label relationship in arcs.

- **Labels**
  States and transitions are labeled with one or more characters, and these are simply treated as text strings. In certain instances, the transition label could be two or more different values separated by a delimiter symbol such as a comma.

Visual marks	Syntactic element	Semantic element
Circle, text	circle with text inside	Normal state
Arrow, circle, text	unlabeled arrow connected to a circle with text inside	Start state
Circle, text	two concentric circles with text inside	Accepting state
Directed line, text	directed line with adjacent text	Transition

Table 5.2: Visual markings mapped to FSA semantics.

5.3.1.3 Semantic elements of FSA diagram notation

In terms of semantics, an FSA diagram instance consists of a *start* state, *accepting* state(s), *normal* state(s), and *transitions*. Between states, there could be one or more transitions. States have names, and the transitions from one state to another have labels. The mapping of these semantic elements from respective visual elements is shown in Table 5.2.

The non-terminal symbol types for an FSA diagram include these semantic elements namely the start state, accepting state(s), normal state(s), and transition(s).

5.4 Formalizing the syntax of FSA diagrams using CMGs

Parsing an FSA diagram requires formalizing the syntax of FSA notation. The preceding sections on FSA diagram notation form a background for writing a CMG grammar for FSA diagrams. To write a grammar, the constructs of the language are first divided into syntactic categories [115]. A syntactic category is a sub-language (in our case a sub-diagram) that embodies a particular concept; examples of syntactic categories in programming languages are expressions, statements, and declarations [115].

For FSA diagrams, the syntactic categories are arcs (an arc in this instance comprises two non-terminals, a directed line terminal symbol and a text terminal symbol), the various types of states and transitions. Each syntactic category is denoted by a main non-terminal, and additional non-terminals may be needed to describe a syntactic category.

5.4.1 Grammar symbol type declarations

Applying CMGs require that all symbol types used in the grammar be stated. Terminal and non-terminal symbol types as well as their attributes are expected to be included in these declarations. For an FSA diagram grammar, the symbol types listed in Table 5.1 are declared.

Additionally, we declare three non-terminal symbol *FSAStates* which are a multiset of all the states in the diagram, *FSATrans*, a multiset of all transitions in the diagram, and *FSA*, the start symbol. The respective data types of the attributes have been left out in these declarations. The data types for most attributes are elementary data types, except for attributes of the *FSAStates, FSATrans,* and *FSA* symbols, which are multisets. In the following declaration the lowercase is used only for style and does not refer to any particular difference to those written in a different case.

declare symboltype arc (startpoint, midpoint, endpoint, label, pairing): nonterminal;
declare symboltype transition (from, to, label): nonterminal;
declare symboltype state (name, midpoint, radius, kind): nonterminal;
declare symboltype fsatrans (): nonterminal;
declare symboltype fsastates (): nonterminal;
declare symboltype fsa (fsastates, fsatrans): starttype;

5.4.1.1 Production for labeled arcs

A labeled arc (or arc for short) *ArcI*, is a multiset containing a directed line *LineJ* and a text *TextK* symbol type, with the condition that *TextK* is adjacent (is the nearest) text element to *LineJ*; the attribute *ArcI.label* of *ArcI* is obtained from *TextK.label* by assignment. Similarly, the attributes *ArcI.startpoint* are obtained from *LineJ.startpoint*, *ArcI.midpoint* from *LineJ.midpoint*, and *ArcI.endpoint* from *LineJ.endpoint* by assignment. An external function *isadjacent* checks whether *TextK* is the closest text element to *LineJ*. In CMG-based grammars, additional functionality which are outside the operations of the associated constraint solver can be introduced. These are handled by external functions such as *isadjacent*.

Our grammar production for an arc therefore is:

$$
\begin{aligned}
ArcI : arc ::= LineJ : directedline, TextK : text \text{ where } (\\
& isadjacent(TextK, LineJ) \\
&){\{} \\
& ArcI.Label = TextK.label; \\
& ArcI.startpoint = LineJ.startpoint; \\
& ArcI.midpoint = LineJ.midpoint; \\
& ArcI.endpoint = LineJ.endpoint; \\
& \}
\end{aligned}
$$

5.4.1.2 Production for normal states

A normal state *StateX* is a multiset containing symbol types circle, *CircleY*, and text *TextZ*; with the condition that *CircleY.midpoint* is at *TextZ.midpoint*. The attributes set of *StateX* (name, midpoint, radius) are derived by assignment, where *StateX.name* is obtained from *TextZ.label*, *StateX.midpoint* from *CircleY.midpoint*, and *StateX.radius* from *CircleY.radius*. The negative constraints feature of the CMG allows to specify a condition making the production valid only if an additional circle sharing the midpoint is not present.

The CMG grammar production for a normal state therefore is:

$$
\begin{aligned}
&StateX : state ::= CircleY : circle, TextZ : text \text{ where } (\\
&\quad CircleY.midpoint == TextZ.midpoint \text{ AND}\\
&\quad \text{not exist } CircleA : circle\\
&\quad \text{where } (CircleA.midpoint == CircleY.midpoint))\\
&\quad)\{\\
&\quad StateX.name = TextZ.label;\\
&\quad StateX.midpoint = CircleY.midpoint;\\
&\quad StateX.radius = CircleY.radius;\\
&\quad StateX.kind = \text{“}normal\text{”};\\
&\quad \}
\end{aligned}
$$

5.4.1.3 Productions for start and accepting states

A start state has all the features of a normal state with the addition of a directed line pointing to it. The line involved does not have a text label attached to it. The negative constraints feature of the CMG allows to specify a condition making the production valid only if a directed line, *LineA*, which does not have a text label in its neighbourhood, exists.

Our CMG grammar production for a start state is:

$$
\begin{aligned}
&StateX : state ::= CircleY : circle, TextZ : text, LineA : directedline \text{ where } (\\
&\quad CircleY.midpoint == TextZ.midpoint\\
&\quad \text{not exist } TextT : text\\
&\quad \text{where } (isadjacent(TextT, LineA))\\
&\quad)\{\\
&\quad StateX.name = TextZ.label;\\
&\quad StateX.midpoint = CircleY.midpoint;\\
&\quad StateX.radius = CircleY.radius;
\end{aligned}
$$

$StateX.kind = \text{“}start\ state\text{”};$
$\}$

In the instance of an accepting state, the presence of an additional circle, *CircleA*, in the vicinity of a state symbol confirms a symbol as an accepting state.

$StateX : state ::= CircleY : circle, CircleA : circle, TextZ : text$ where (
$CircleY.midpoint == TextZ.midpoint$
$CircleA.midpoint == TextZ.midpoint$
$CircleY.radius > CircleA.radius$
$)\{$
$StateX.name = TextZ.label;$
$StateX.midpoint = CircleY.midpoint;$
$StateX.radius = CircleY.radius;$
$StateX.kind = \text{“}accepting\ state\text{”};$
$\}$

5.4.1.4 Production for transitions

A transition, *TransJ*, is a multiset containing states *StateX* and *StateY*, and an arc, *ArcZ*, such that *ArcZ* is paired with *StateX*, and also paired with *StateY*. Evaluating this condition consequently involves some additional computation.

This particular production case for transitions requires an external function *ispaired* which returns a boolean value that either confirms or rejects the test. The parameters of *ispaired* are the pairing data for *ArcZ*, and the bounding box coordinates of *StateX* and *StateY*.

Furthermore, they utilize existential quantification, since we do not want states *StateX* and *StateY* reduced when the production is applied. The grammar production for the FSA diagram transition non-terminal therefore is:

$TransJ : transition ::= ArcZ : arc$ where (
exist $StateX : state, StateY : state$ where (
$ispaired(ArcZ.pairing, StateX.boundingbox, StateY.boundingbox)$
$)\{$
$TransJ.from = StateX.name;$
$TransJ.to = StateY.name;$

$TransJ.label = ArcZ.label;$
$\}$

5.4.1.5 Production for the grammar start symbol

The start symbol *FsaX* consists of a multiset of states and transitions; essentially an FSA diagram is a collection of states and transitions. The CMG formalism includes features which allow all sub-diagrams of the same type to be collected. Like in the other productions, constraints can be added to restrict membership to certain types of sub-diagrams only [29]. All symbols of type states are collected into *FSAStatesX* and all symbols of type transition are collected into *FSATransY*.

$$FsaX : fsa ::= allStateA : state, allTransB : transition \text{ where } (true)$$
$$\{$$
$$FSAStatesX = \{StateX.name\};$$
$$FSATransY = \{TransY.from, TransY.to, TransY.label\};$$
$$\}$$

By the production collecting recognized states and transitions (as multisets) into the start symbol, the 'read' or 'interpreted' diagram is obtainable, since the different primitive elements of the FSA diagram have been grouped into various semantic elements according to the syntax of FSA notations.

The grammar described in this chapter and the output from the diagram image analysis stage (discussed in Chapter 3), are key inputs for a parsing procedure which will use the grammar to interpret the primitives and their spatial arrangements. In Chapter 6, each stage of our recognition system is tested by experiments and the results reported, including an overview of how this FSA grammar will be applied in the interpretation of FSA diagrams.

Chapter 6

Experiments and results

"Test all things, hold on fast to that which is good" (1 Thes. 5:21, King James Version).

This thesis concentrates on the automatic extraction of domain specific meaning from a diagram image, in particular FSA diagrams. While our concern is biased towards the high level interpretation of diagram semantics, such level of interpretation, however, is dependent on first obtaining the concrete visual and spatial elements of the diagram.

Since a tool to satisfactorily extract such information was not readily available, and attempting to use standard commercial OCR software failed, the first part of this thesis examined the design of a graphics recognition system for extracting diagram contents needed to interpret diagrams. The system operates sequentially, with output from one phase serving as input into the next phase.

The major phases of our graphics recognition system which is presented visually in Figure 6.1 are skeletonization, text-graphics separation, graphics decomposition (node-link separation), graphics categorization, and the semantic interpretation phase. In this chapter, the operation of these phases is examined.

6.1 Testing the recognition system

The multiphase nature of diagram recognition systems informs experimentation on a phase-by-phase basis. In carrying out these tests, the qualitative effectiveness of each phase is tested and reported. The output at that stage is evaluated and thereafter, testing proceeds to the next phase.

The input to each phase will be an ideal output instance obtained from the previous phase. For example, the text-graphics separation phase should output, amongst others, a graphic image layer containing graphics elements only. This output is the expected input to the graphics decomposition phase.

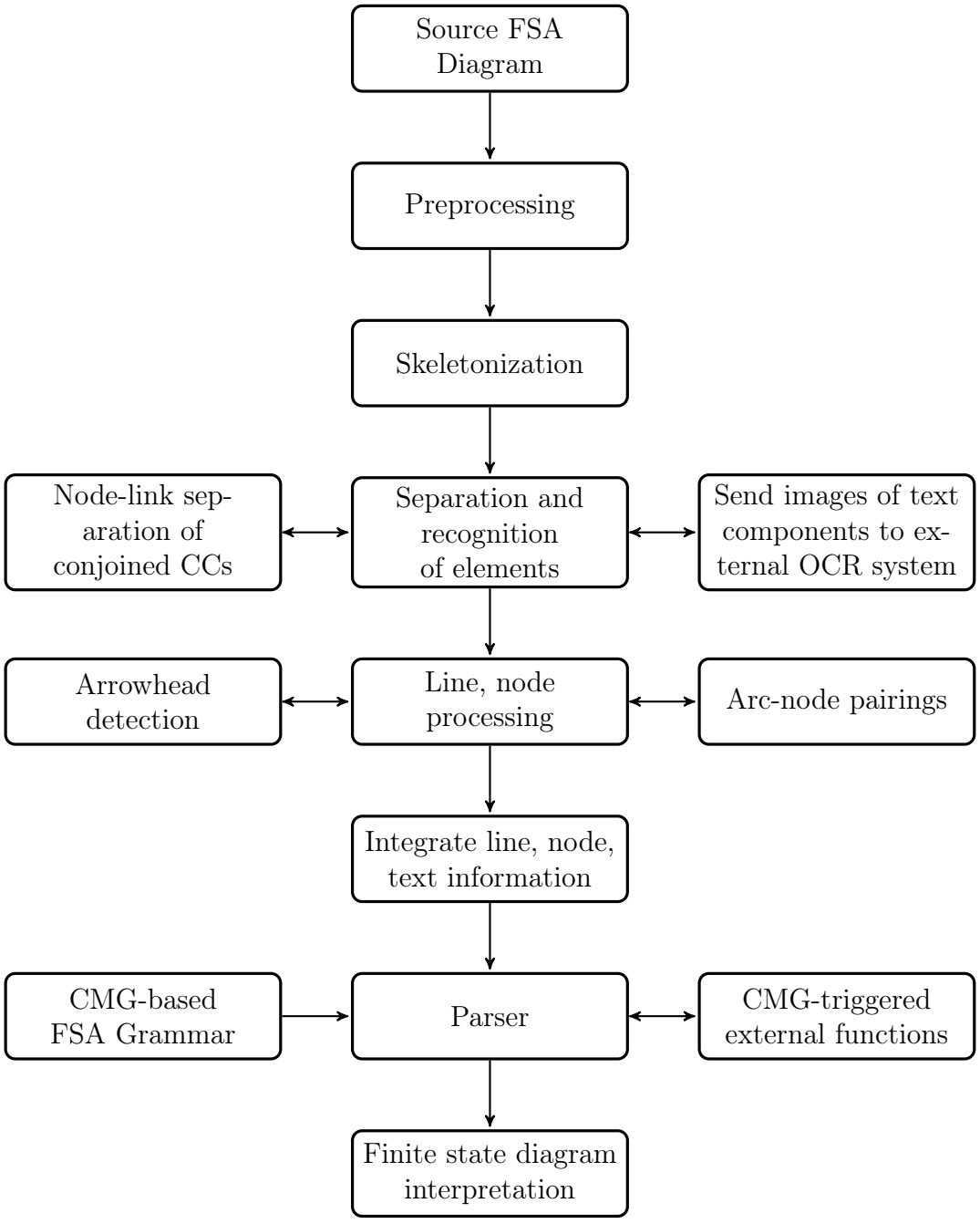


Figure 6.1: The FSA diagram recognition scheme.
(CC–Connected component. OCR–Optical Character Recognition).

If the actual output is less than ideal, for instance in the case of text characters not completely removed from the graphics, an ideal output image (manually corrected) will be used as input to test the next phase of the diagram recognition process. The individual strengths and weaknesses of various phases of the system is thereby discovered and clearly analysed, allowing more of the challenges in graphics recognition to be seen, and showing areas requiring improvement and further research.

6.1.1 Testing setup

The system was implemented using the Enthought Python Distribution [1] of the Python programming language. The distribution provides a comprehensive environment for scientific computing and makes it easy to access and manage Python packages such as the Python Image Library (PIL), Numpy, and scikit-image, among over 250 scientific and analytic Python packages. These contain some essential libraries for implementing image and computer vision processing in Python. An advantage of these libraries in terms of availability is the fact that they are open-source.

The scikit-image image processing library [149] version 0.10.1, includes image processing modules such as Feature, Measure, and Morphology which were useful in implementing this recognition system.

The development and testing hardware is a 32-bit, 2.40GHz Intel Core i5 processor computer with 4GB random access memory (RAM), running the Windows 7 operating system.

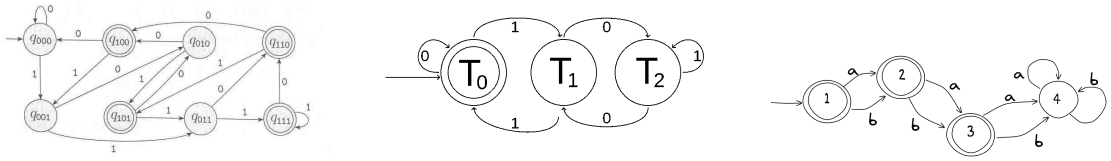
6.1.1.1 Diagrams used in the experiments

Real-life diagrams were obtained, each representing different looks of instructional diagrams. Three of the test diagrams shown in Figure 6.2 respectively will be used to illustrate and analyse the various parts of our recognition system in detail.

The variety of possible styles used in the production of diagrams is the first observation we point out. The diagrams also vary in quality and content. Figure 6.2 shows thumbnails of the test diagrams. Larger sizes of the test diagrams are shown in Figures 6.3, 6.4, and 6.5.

The major differences between the images include positioning of arc arrowheads relative to the respective circles they are directed at, and stroke styles. In the case of arrowheads, they could either touch the circles as is the case in *Fsa1* (Figure 6.3) or could have a gap between the arrowhead and circle border as in *Fsa2* (Figure 6.4). These properties potentially influence the diagram recognition process and had to be considered in designing the diagram recognition system.

A number of limitations exists with the recognition stage of the interpretation system. In non-planar graphs, the connecting lines cross at some points. This



(a) *Fsa1.* (b) *Fsa2.* (c) *Fsa3.*

Figure 6.2: Thumbnails of some diagrams used in experiments.

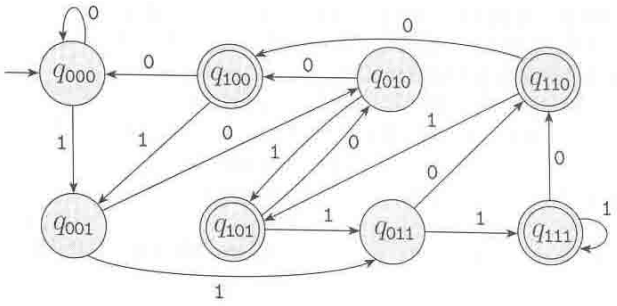


Figure 6.3: *Fsa1.* An FSA diagram drawn with lines touching circles, taken from [138].

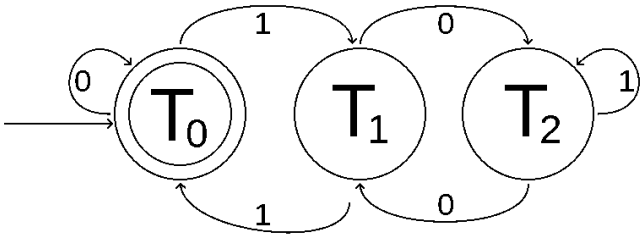


Figure 6.4: *Fsa2.* An FSA diagram drawn with lines detached from nodes.

condition is not currently catered for in the structural recognition phase as the functionality requires additional processing. This was dealt with in [6]. However, the semantic interpretation phase is not impacted, since there is no syntactic denotation attached to crossing lines in the FSA diagram notation.

Similarly, if arc labels are positioned such that they break the line into separate segments, it leads to erroneous recognition as the condition is currently not addressed by the recognition system.

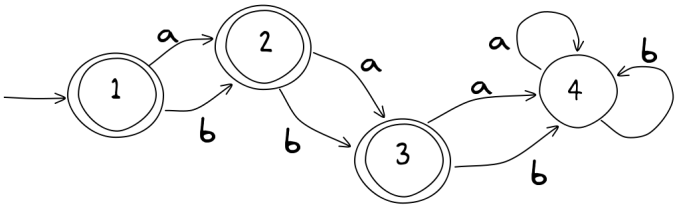


Figure 6.5: *Fsa3.* A hand-drawn FSA diagram with most directed lines detached from the nodes [141].

6.1.2 Sequence of operations and the resulting output

For each diagram tested, a series of outputs are obtained. The output is either graphics, or a combination of numeric and alphanumeric values. The flow of operations in the recognition system is presented in Figure 6.1. The skeletonization, text-graphics separation, element categorization, and arrowhead recognition processes produce images as output.

The pairing process, which assigns to an arc a proximally located circle at each of the two ends, produces alphanumeric results. This process lists the arc coordinates, and the node coordinates associated with each directed line. The archead detection process detects the target end of arcs in the diagram, and reports which segment of the directed line bears the arrowhead.

The line and node processing stage produces a summary of the various elements detected and what class of elements they belong to. An additional class named *miscellaneous* designates the category of elements the recognition system is unable to identify.

6.2 Recognition tests and results

The input diagrams are either existing images from electronic documents or diagrams scanned from textbooks at varying resolutions and dimensions. After acquiring the image, applying thresholding, and inverting it (changing background pixels from white to black and the foreground pixels from black to white), it is saved as a monochrome bitmap image. The inverted image is then skeletonized, separated into different layers, and the elements identified. In this section, these processes are tested and the results reported. The reader may note that all images are shown in the non-inverted form, for legibility reasons.

6.2.1 Skeletonization results

The respective skeleton images for the diagrams are shown in Figures 6.6, 6.7, and 6.8. While the skeleton images produced by the skeletonization process maintained

connectivity of contours, junction patterns, especially those at the boundaries of circles, exhibit significant variations.

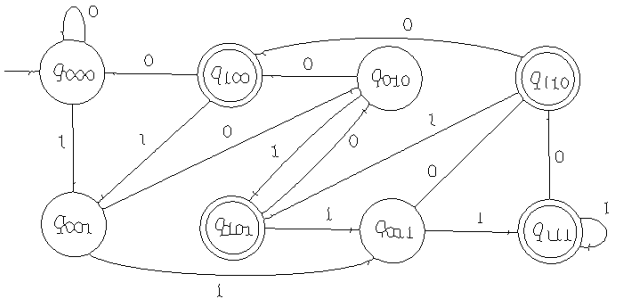


Figure 6.6: Skeleton image for *Fsa1*.

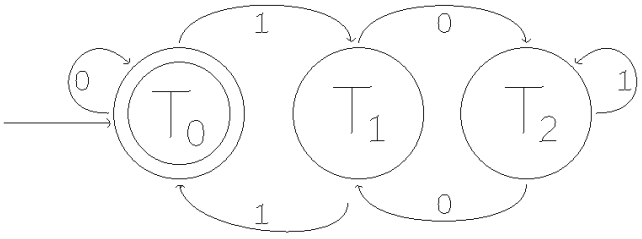


Figure 6.7: Skeleton image for *Fsa2*.

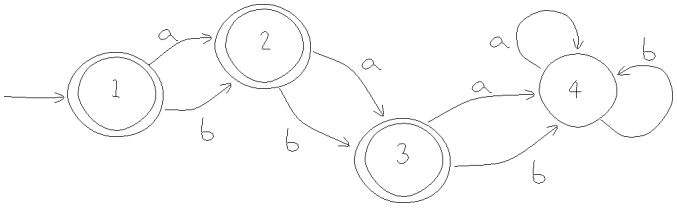


Figure 6.8: Skeleton image for *Fsa3*.

Almost every junction region in the skeleton image has its own unique pattern. The images in Figure 6.9 show four different pixel arrangements for the same 270° junction type from *Fsa1*.

It can be observed from the images that junction patterns vary within images and also across different images. This situation is a principal source of challenge to the node-link separation process as will be seen later.

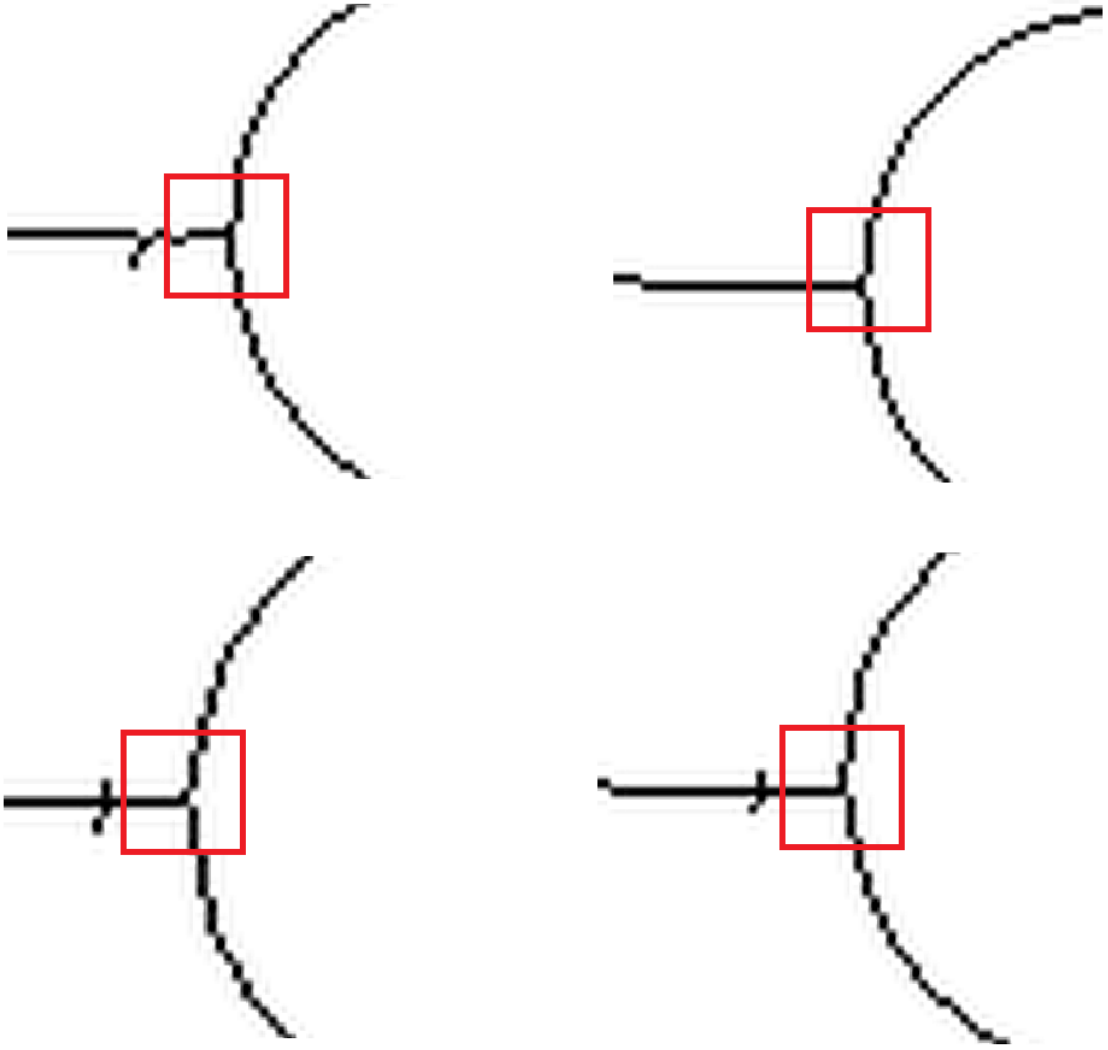


Figure 6.9: Segments of *Fsa1* show a set of similar junction types, but with each having a different pixel pattern. The junction areas are highlighted with red outlines.

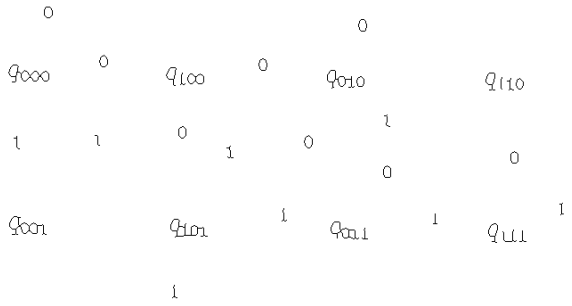


Figure 6.10: Text layer for *Fsa1*.

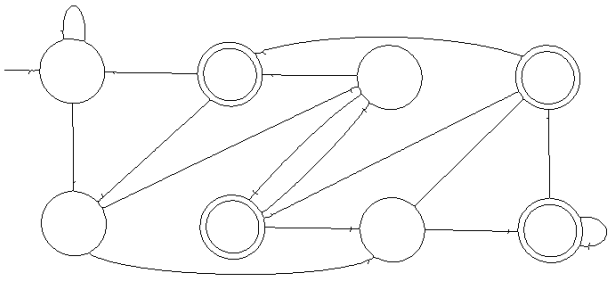


Figure 6.11: Graphics layer for *Fsa1*.

6.2.2 Text-graphics separation results

Text-graphics segmentation results for test diagrams are shown in Figures 6.10, 6.12, and 6.14. The respective graphics layers remaining after the separation process are shown in Figures 6.11, 6.13, and 6.15.

The results show that all text characters were successfully extracted from the diagrams in the test set. The text layer for *Fsa1* (Figure 6.10) shows the problem of merged text (see for example the label at the bottom right of Figure 6.10). The merged text occurred after skeletonization was carried out on the image, and it was due to the closeness of the text characters in the diagram. Therefore, for the actual character recognition, using regions extracted from the original diagram may be a better choice. Since the position of text is established by the text-graphics separation, the text region can be directly extracted from the original non-skeletonized image.

6.2.3 Graphics decomposition (node-link separation)

The identification of every symbol in the diagram is necessary for its correct interpretation. If one element is not correctly detected as a result of incomplete separation, it affects the detection of the conjoined partner and overall, the inter-

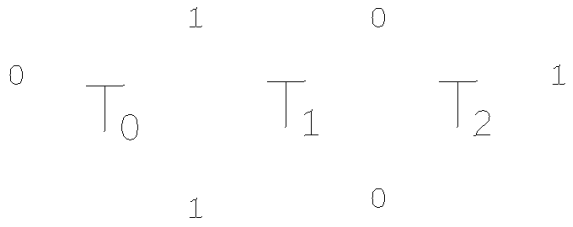


Figure 6.12: Text layer for *Fsa2*.

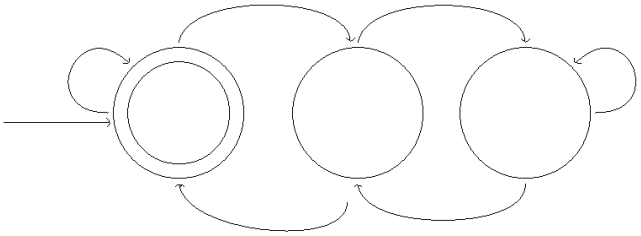


Figure 6.13: Graphics layer for *Fsa2*.

Figure 6.14: Text layer for *Fsa3*.

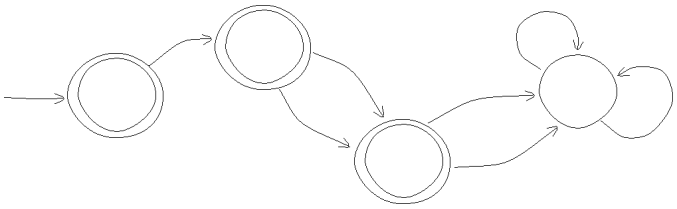


Figure 6.15: Graphics layer for *Fsa3*.

pretation of the diagram. Nodes are important elements in FSA diagrams and all nodes must be located and separated from connecting lines so that they can be identified.

The junction identification process as earlier described in Section 3.3.3.2, detects all junction points but has challenges with distinguishing which pixels in the junction vicinity lie on the node borders and which are part of the connection line. This deficiency is mainly due to the unpredictable junction patterns. This leads to failure of the node-link separation in some cases. Junctions where the separation process failed can be seen in Figure 6.17, which are highlighted in Figure 6.18. This problem is exacerbated by the fact that the attempt to identify pixels belonging to the line element is based on analysing local segments over an area covering only 3×3 pixels.

Increasing the size of the window area to a 5×5 pixel area for example does not yield an improvement in the instances we examined. This is because the conditions challenging the determination of line pixels from node pixels are still present (unpredictable junction patterns and local analysis) even though the area increases. Line analysis (for instance line following) is a possible solution however it is not a local operation.

The disturbed nature of pixels at the junctions in skeleton images of the diagrams also raise major issues. The first issue is the attendant complication in detecting junctions. Identifying pixels of the connecting line at a junction becomes complicated due to the range of possible patterns that must be searched for. The set of possible patterns at junctions far exceeds the number of model junction patterns (lines joining node borders at 45°, 90°, 135°, 180°, 225°, 270°, 315°, and 360°). Secondly, the node outline structures are not smooth, nor are they always regularly patterned.

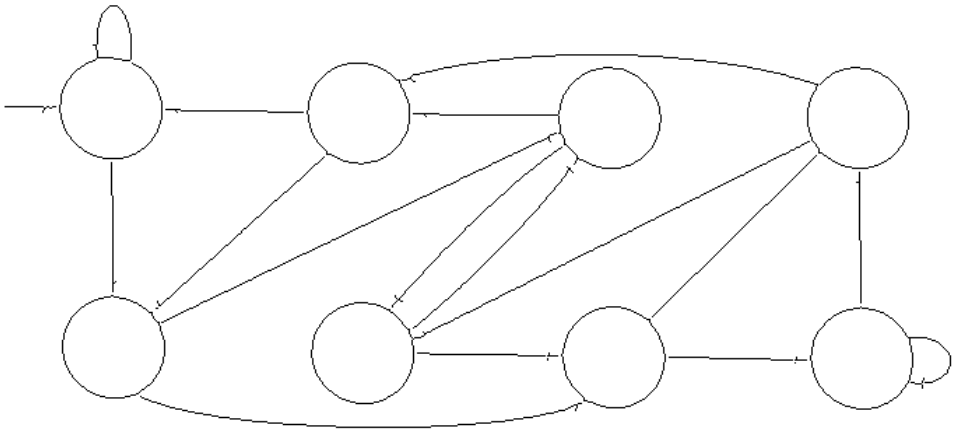


Figure 6.16: *Fsa1* node-link separation result.

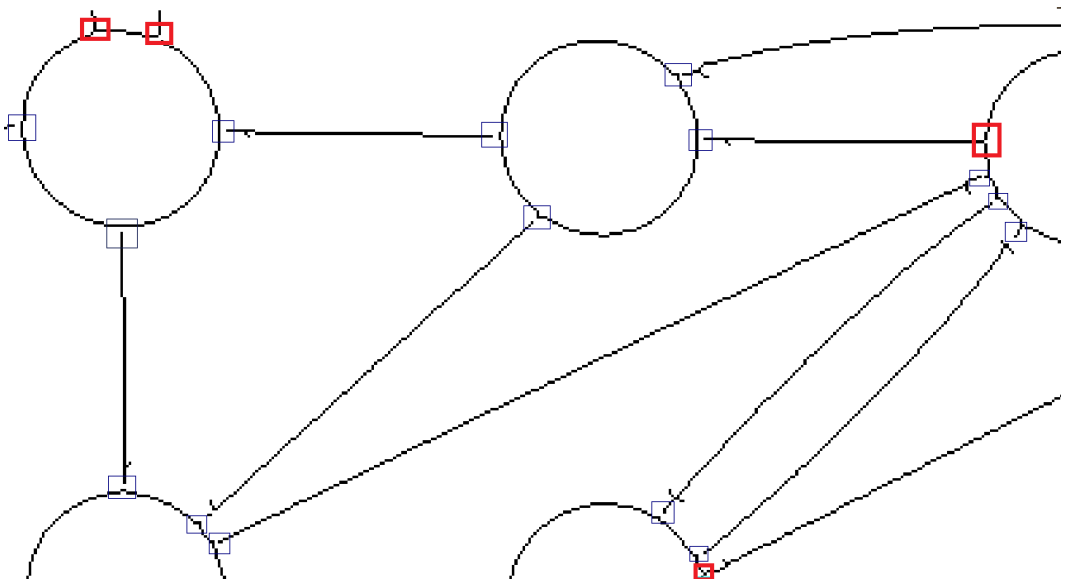


Figure 6.17: *Fsa1* unsevered junctions are marked by red outlines in this diagram.

Our solution to failed node-link separation is the unconditional disconnection of all points that are potential node-link junctions. While this operation leads to fragmentation of the symbols affected, the benefit is that all junction points in the image are detected and severed. The disadvantage is that node borders become fragmented.

The unconditional disconnection of junctions is the final step after all model junction patterns may have been searched for and an attempt made to disconnect the incident line from the symbol border. The results for the unconditional disconnection operation show that all the junctions, as well as arrow heads, are separated by the process. The image in Figure 6.16 shows the line-symbol disconnection result of *Fsa1*, and Figures 6.17 and 6.18 show enlarged segments of the diagram and highlights where node-link separation failed. Figure 6.19 shows the unconditional disconnection results for the diagram.

The process of identifying circles and lines in the image succeeds when the node is not fragmented due to a broken boundary. However, if node boundaries are broken due to the disconnection of junctions, an additional process is required to reconstruct the borders from the border fragments. Therefore, a special reconstruction operation is applied on node elements after the unconditional disconnection operation is applied. Some research on the reconstruction of contours and lines are found in [81, 133, 153].

The automatic reconstruction process which must follow the unconditional disconnection process is not yet implemented, and is left to further research. The manual reconstruction process of *Fsa1* results in the image shown in Figure 6.20. This reconstructed version of *Fsa1* is used as input for the next phase.

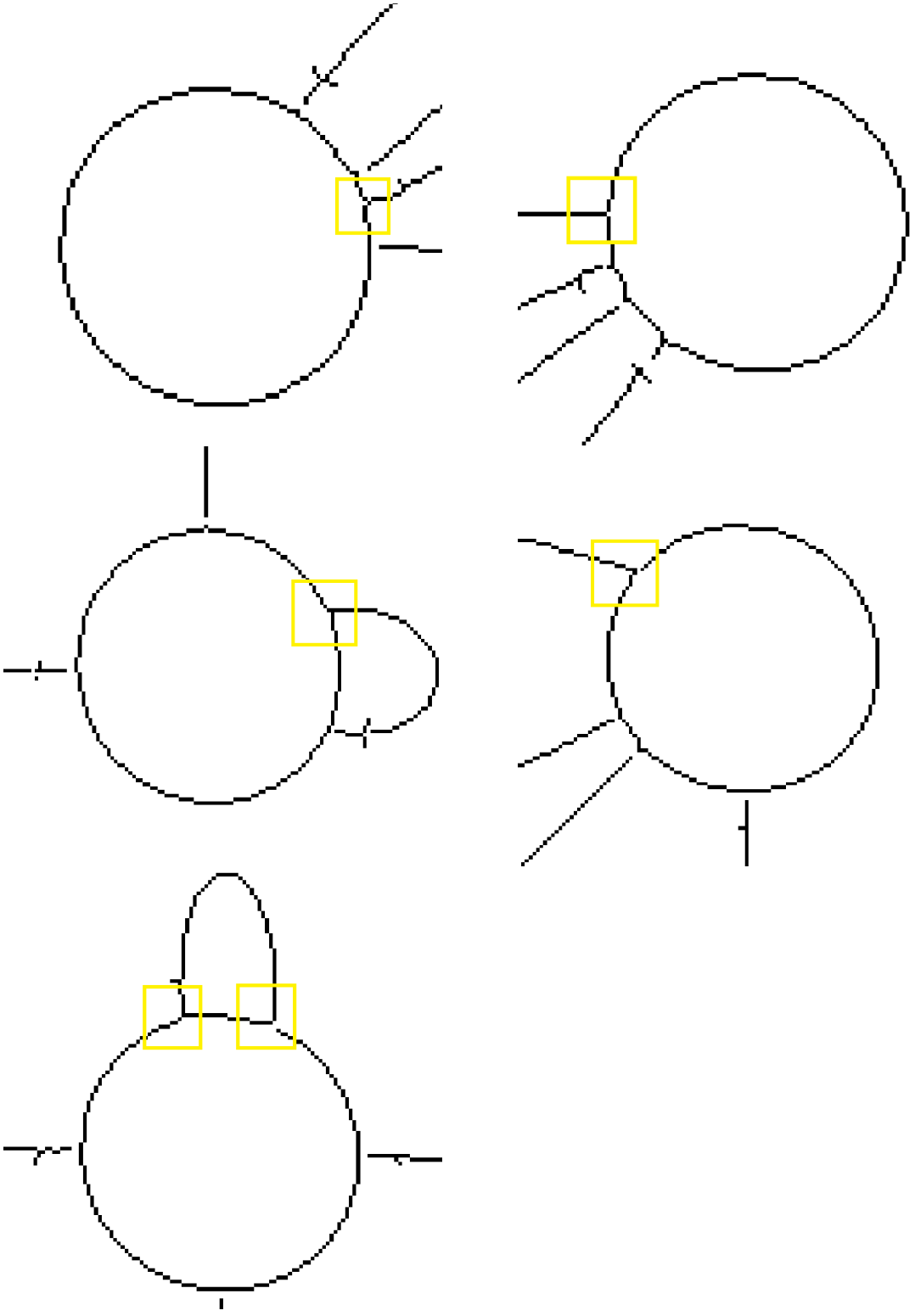


Figure 6.18: Unsevered junctions in *Fsa1* highlighted with yellow outlines.

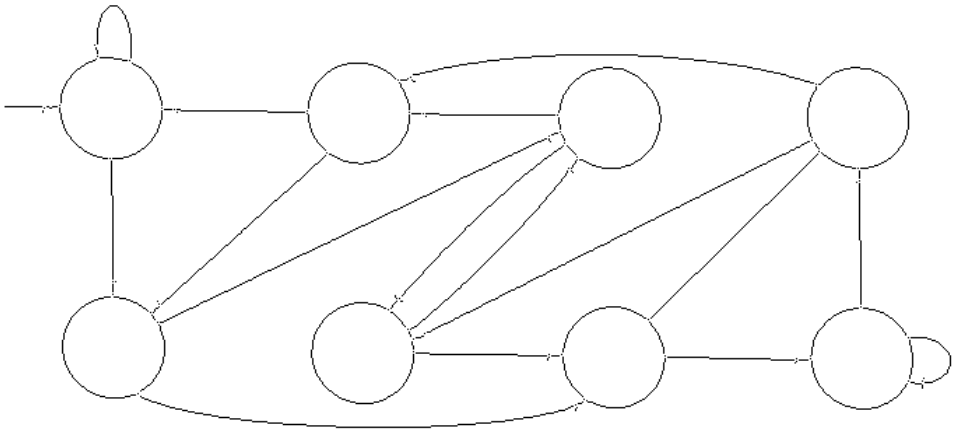


Figure 6.19: Unconditional decomposition of junctions in *Fsa1*.

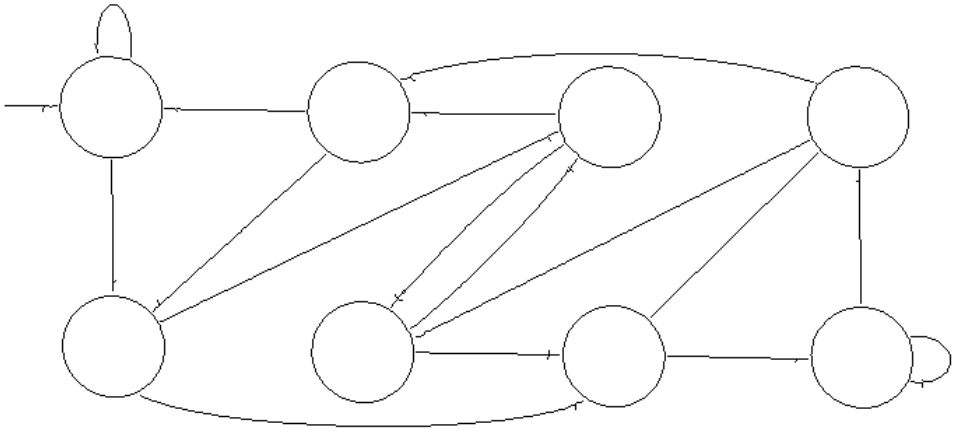


Figure 6.20: *Fsa1* reconstructed after unconditional decomposition.

6.2.4 Categorization of diagram components

The symbol recognition phase undertakes classification of the remaining graphics into either lines or circles. Figures 6.21, 6.23, and 6.24 show the objects classified as circles, and Figures 6.25, 6.26 and 6.27 show the objects classified as lines. Comparing these images with the related *miscellaneous* category image (Figure 6.30) shows which elements in the diagram are undetected. Overall, it is conjoined elements that remain undetected. It is noteworthy that all isolated node elements in the diagrams *Fsa1, Fsa2, and Fsa3* are identified. For *Fsa2*, no undetected element remained.

Figure 6.21: Elements categorized as circles in *Fsa1* before node-link separation. Only the inner circles of the concentric circle representing accepting states are recognized.

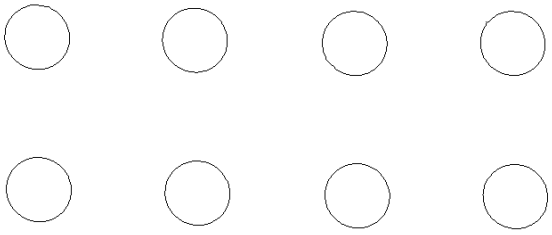


Figure 6.22: Additional elements categorized as circles in *Fsa1* after node-link separation. All nodes are recognized.

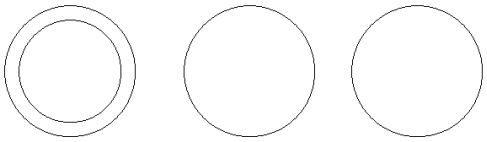


Figure 6.23: Elements categorized as circles in *Fsa2*.

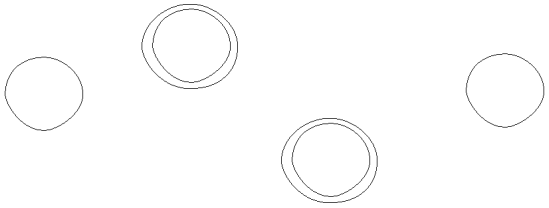


Figure 6.24: Elements categorized as circles in *Fsa3*.

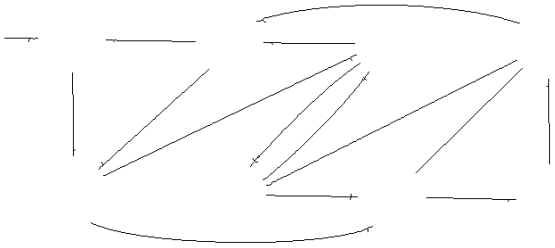


Figure 6.25: Arc elements in *Fsa1*.

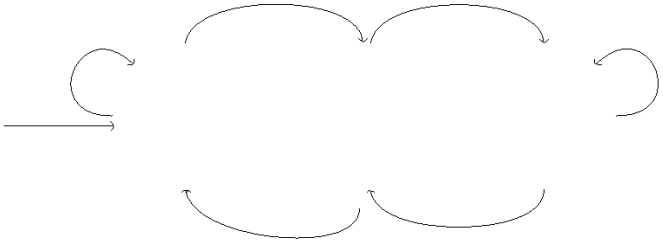


Figure 6.26: Arc elements in *Fsa2*.

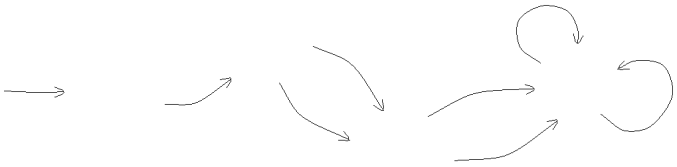


Figure 6.27: Arc elements in *Fsa3*.

The miscellaneous layer contains unrecognized diagram elements. As earlier noted, the likely elements in this layer are unsegmented objects in the diagram. Figures 6.28 and 6.30 show the miscellaneous elements in the diagrams.

The two elements recognised as text for *Fsa1* (after reconstrution) are actually arcs. In the original diagram with text, the presence of text in the image results in lower median value. But since there are no text characters in the image when it goes through the second recognition attempt, the median value becomes high and smaller objects having text like features such as the small arcs (see 6.29) are categorised as text. This will be rectified in future work. A possible solution is to ensure that values from the first round of recognition are stored and maintained for use on subsequent rounds of the recognition process. After reconstruction, the orginal median and mean values are used as parameters; in that way, new values which will cause such errors are avoided.

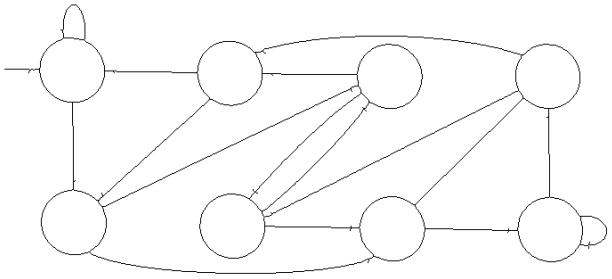


Figure 6.28: Miscellaneous layer for *Fsa1* before node-link separation. After separation, no element remain undetected.

Figure 6.29: Elements classified as text in *Fsa1* after node-link separation. The identification of these arcs failed because they were smaller than the median size of objects in the separated diagram.

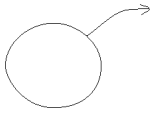


Figure 6.30: Miscellaneous layer for *Fsa3*. The conjoined elements resulted in non-recognition.

Arc	Bounding box coordinates
Arc0	58, 307, 79, 603
Arc1	95, 24, 100, 62
Arc2	97, 138, 100, 239
Arc3	100, 315, 103, 418
Arc4	114, 135, 250, 420
Arc5	120, 318, 261, 600
Arc6	123, 300, 240, 424
Arc7	129, 486, 247, 606
Arc8	130, 130, 243, 254
Arc9	133, 314, 255, 434
Arc10	134, 100, 228, 104
Arc11	141, 633, 237, 637
Arc12	270, 318, 277, 421
Arc13	274, 498, 279, 599
Arc14	302, 121, 325, 438

Table 6.1: Bounding box coordinates for detected arcs in *Fsa1*.

Node	Bounding box coordinates
Circle0	60, 63, 133, 137
Circle1	64, 240, 137, 314
Circle2	67, 420, 140, 494
Circle3	67, 598, 140, 671
Circle4	231, 65, 304, 139
Circle5	235, 243, 308, 317
Circle6	238, 423, 311, 497
Circle7	239, 600, 312, 674

Table 6.2: Bounding box coordinates for detected nodes in *Fsa1*.

6.2.5 Pairing connecting lines (arcs) with nodes

Each arc in the image has a relationship with a source and a target node (except the arc symbolizing the start state). The purpose of the pairing operation is to identify the nodes to which each arc is linked. However, it does not identify which is the source node and which is the target node. A later arrowhead recognition operation does that.

Figure 6.25 shows the set of arcs detected in *Fsa1*, and the results of the pairing process for $Fsa1$ is produced in Table 6.3. The results of the pairing stage show that the operation successfully identifies all the circles to which each line is related.

Arc-node pairing summary
Arc 0 ending at column 603 pairs Circle 1 ending at column 314
Arc 0 ending at column 603 pairs Circle 3 ending at column 671
Arc 1 ending at column 62 pairs Circle 0 ending at column 137
Arc 2 ending at column 239 pairs Circle 0 ending at column 137
Arc 2 ending at column 239 pairs Circle 0 ending at column 137
Arc 2 ending at column 239 pairs Circle 1 ending at column 314
Arc 3 ending at column 418 pairs Circle 1 ending at column 314
Arc 3 ending at column 418 pairs Circle 1 ending at column 314
Arc 3 ending at column 418 pairs Circle 2 ending at column 494
Arc 4 ending at column 420 pairs Circle 4 ending at column 139
Arc 4 ending at column 420 pairs Circle 6 ending at column 497
Arc 5 ending at column 600 pairs Circle 5 ending at column 317
Arc 5 ending at column 600 pairs Circle 5 ending at column 317
Arc 5 ending at column 600 pairs Circle 7 ending at column 674
Arc 6 ending at column 424 pairs Circle 5 ending at column 317
Arc 6 ending at column 424 pairs Circle 6 ending at column 497
Arc 7 ending at column 606 pairs Circle 6 ending at column 497
Arc 7 ending at column 606 pairs Circle 7 ending at column 674
Arc 8 ending at column 254 pairs Circle 4 ending at column 139
Arc 8 ending at column 254 pairs Circle 5 ending at column 317
Arc 9 ending at column 434 pairs Circle 5 ending at column 317
Arc 9 ending at column 434 pairs Circle 6 ending at column 497
Arc 12 ending at column 421 pairs Circle 5 ending at column 317
Arc 12 ending at column 421 pairs Circle 5 ending at column 317
Arc 12 ending at column 421 pairs Circle 6 ending at column 497
Arc 13 ending at column 599 pairs Circle 6 ending at column 497
Arc 13 ending at column 599 pairs Circle 6 ending at column 497
Arc 13 ending at column 599 pairs Circle 7 ending at column 674
Arc 14 ending at column 438 pairs Circle 4 ending at column 139
Arc 14 ending at column 438 pairs Circle 6 ending at column 497

Table 6.3: Arc-node pairing for *Fsa1*. The information is presented only as a coarse illustration of the underlying process.

6.2.6 Results of the arrowhead detection process

The source and target points of directed lines must also be identified. The identification of these relationships is based on the arrowheads of the arcs. The arrowhead detection process is based on the assumption that arrowhead regions have more foreground pixels than other portions of the line.

The results of the arrowhead detection in test FSA diagrams are shown in the images in Figures 6.31, 6.32 and 6.33. The region detected as bearing the arrowhead is highlighted with a black block.

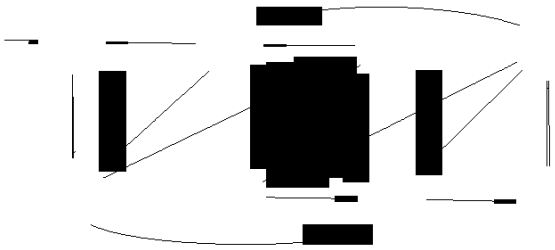


Figure 6.31: *Fsa1* arrowhead detection results. The black blocks mark the sections of the arc bearing the arrowhead; whether the leftmost or the rightmost segment of the arc. The markings overlap due to the high density of the arcs in this particular case.

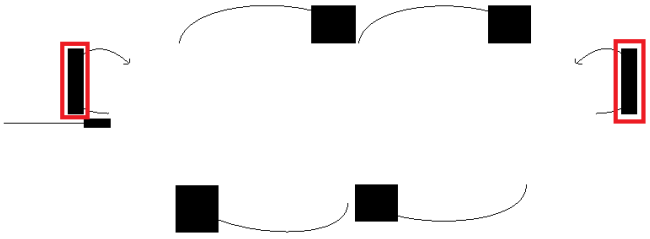


Figure 6.32: *Fsa2* arrowhead detection results. Red outlines highlight wrong detection of arrowheads in looped arcs.

In looped arcs, the end carrying the arrowhead will likely always have a lower number of pixels, this is due to the fact that one side of looped arcs is a somewhat open section as seen from the area labeled Z_1 in Figure 6.34. The arrowhead detection process examines the two extreme sections of the arc (*Zones* Z_1 *and* Z_4) seeking for the zone with a higher foreground pixel count. Hence, arrowheads in looped directed lines - such as those highlighted with additional red outline in Figures 6.32 and 6.33 are not correctly detected. However, this error does not

impact the interpretation, since loops have their source and target pointing at the same node.

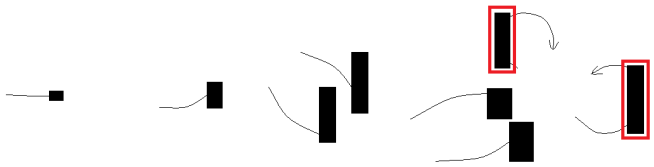


Figure 6.33: *Fsa3* arrowhead detection results. Red outlines highlight wrong detection of arrowheads in looped arcs.

The special directed arc in FSA diagrams marking the start state can be directly identified in the image by searching for an arc with one 'unbounded' side. However, this is a notation-specific detail, and is left to the semantic interpretation stages.

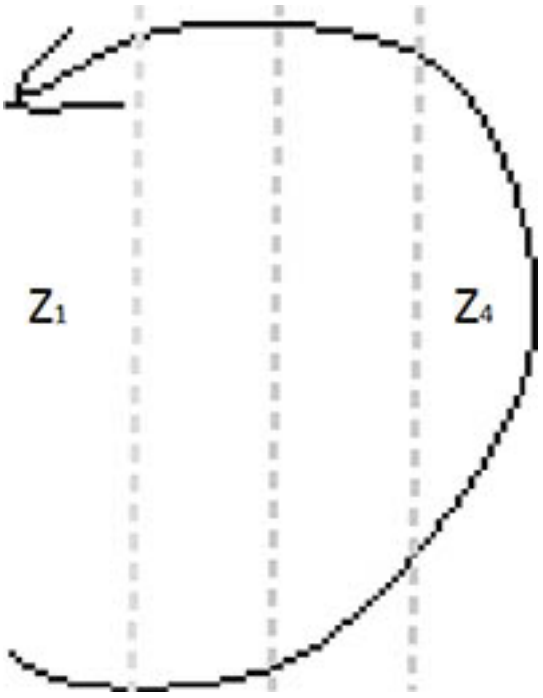


Figure 6.34: Looped arc divided into four zones. Zone Z_1 has fewer foreground pixels than Z_4. This causes failure of the arrowhead detection process when applied to looped arcs.

6.3 Spatial parsing process of a typical FSA diagram based on our FSA grammar

The recognition and interpretation process for an FSA diagram was illustrated in Figure 6.1. The bottom-up parser parses the diagram according to the CMG grammar, with the set of rules presented in Figure 6.35; the diagram's primitives as well as their attributes have been extracted as described in an earlier section, and stored in a database. These rules have been discussed in Chapter 5. The grammar and the database are input into the parser.

Tables 6.4, 6.5, 6.6, and 6.7 show the actual information extracted from *Fsa2*, as stored in the database to be passed to the parsing procedure. The information include centroid coordinates, bounding box coordinates, and pairing information for directed lines detected in the diagram. Without loss of generality, and due to repetitiveness, we continue our example only with *Fsa2*.

```
Rule 1:
ArcI:arc ::= LineJ:directedline, TextK:Text where (
   isadjacent(TextK,LineJ)
) {
   ArcI.Label = TextK.label;
   ArcI.startpoint = LineJ.startpoint;
   ArcI.midpoint = LineJ.midpoint;
   ArcI.endpoint = LineJ.endpoint;
   ArcI.pairing = LineJ.pairing;
  }

Rule 2:
StateX:state ::= CircleY:circle, TextZ:Text where (
CircleY.midpoint == TextZ.midpoint AND
not exist CircleZ:circle
             where  (CircleZ.midpoint == CircleY.midpoint)
) {
   StateX.name = TextZ.label;
   StateX.midpoint = CircleY.midpoint;
   StateX.radius = CircleY.radius;
   StateX.kind = "normal";
   }

Rule 3:
StateX:state ::= CircleY:circle, CircleA:circle, TextZ:Text where (
CircleY.midpoint == TextZ.midpoint
CircleA.midpoint == TextZ.midpoint
```

```
CircleY.radius > CircleA.radius
) {
   StateX.name = TextZ.label;
   StateX.midpoint = CircleY.midpoint;
   StateX.radius = CircleY.radius;
   StateX.kind = "accepting state";
   }


Rule 4:
StateX:state ::= CircleY:circle, TextZ:Text, LineA:directedline where (
  CircleY.midpoint == TextZ.midpoint) AND
      not exist TextT:text
      where (isadjacent(TextT, LineA)) {
   StateX.name = TextZ.label;
   StateXboundingbox = CircleY.boundingbox;
   StateX.radius = CircleY.radius;
   StateX.kind = "start state";
     }

Rule 5:
TransJ: transition ::=  ArcZ:arc  where  (
    exist StateX:state, StateY:state where (
    ispaired(ArcZ.pairing, StateX.boundingbox, StateY.boundingbox)  )
){
  TransJ.from = StateX.name;
  TransJ.to = StateY.name;
  TransJ.label = ArcZ.label;
}

Rule 6:
FsaX:fsa ::= all StateA:state, all TransB:transition where (true)
            {
  FSAStatesX = {StateX.name };
  FSATransY = {TransY.from, TransY.to, TransY.label };
  }
```

Figure 6.35: FSA Grammar Production Rules.

In Figure 6.36, the highlighted areas show groups of primitives which are reduced into non-terminal symbols based on *Rule 1*. The rule reduces directed lines with adjacent text primitives to type *Arc*. These newly created objects are added to the

`ParseForest`. The `ParseForest` is a data structure holding the multiset of parse trees.

The attributes of *Arc* objects are inherited from those of the terminal symbols involved in the reduction. *Rule 1* reduces the primitives highlighted, introducing six new non-terminal objects into the `ParseForest`. These are highlighted in Figure 6.36.

Directed lines *1-7* of Table 6.4, text elements *8-11,* and *18-19* of Table 6.5 are candidate primitives for *Rule 1*. By comparing the bounding box information for the directed lines and text elements from Tables 6.4 and 6.5, the parser is able to find the closest text element to each of the directed lines present in the diagram. From the tables it can be seen that *Element 1* has bounding box x extents between pixel location 154 and 427, and *Element 8* has its x extents between pixel position 163 and 187, making it the text element nearest to *Element 1* in the x direction.

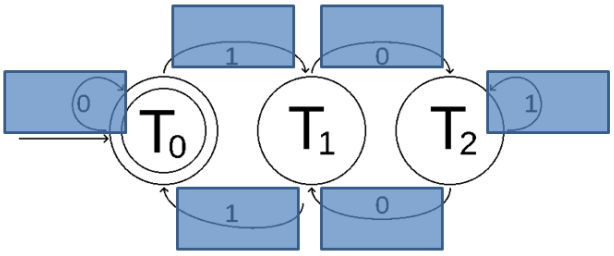


Figure 6.36: Each individual group of highlighted elements is reduced to a labeled arc.

Rule 2 searches for circle primitives with text inside them, with the condition that there is no inner circle. Three such circles matching this rule are found in the database. The individual circles and their text label primitives are reduced to non-terminals of type *normal state*, and added to the `ParseForest`. In Figure 6.37, the three states and existing non-terminals are highlighted.

Some extracted information about the circle elements in the diagram is contained in Table 6.6. Using this information, the parser searches the database, and can apply *Rule 2* to process circles *21-22* and text elements *15-16*. The circle *Element 20* and *Element 23* does not fit the conditions in the rules because two circles enclose the text *Element 17*.

In a similar fashion, *Rule 3* searches for circles. The circles that are left will be those which have not been reduced by *Rule 2*. The rule confirms an accepting state by the presence of a state within the circle. In Figure 6.38, the state recognized as an accepting state is indicated by the dotted arrow.

The next rule (*Rule 4*) searches for a directed line. If found, and under the conditions that the line has only one side paired to a state, the state is assigned as

ID	Centroid coordinates	Bounding box coordinates
1	323, 166	154, 217, 199, 427
2	534, 167	154, 430, 199, 635
3	113, 237	205, 85, 283, 159
4	731, 237	205, 687, 283, 761
5	76, 293	288, 9, 299, 136
6	527, 396	366, 426, 410, 630
7	312, 406	367, 213, 423, 418

Table 6.4: Directed lines information extracted from *Fsa2*.

ID	Centroid coordinates	Bounding box coordinates
8	314, 176	163, 308, 187, 323
9	531, 174	164, 524, 187, 540
10	745, 244	231, 739, 255, 754
11	101, 242	232, 94, 255, 110
12	422, 265	250, 400, 306, 446
13	626, 265	250, 604, 306, 650
14	207, 270	255, 185, 311, 231
15	451, 301	284, 444, 316, 461
16	655, 301	285, 645, 316, 665
17	236, 304	290, 227, 321, 247
18	531, 388	378, 524, 401, 540
19	314, 402	389, 308, 413, 323

Table 6.5: Text character locations extracted from *Fsa2* diagram.

ID	Centroid coordinates	Bounding box coordinates
20	216.8977273, 281	204, 139, 360, 295
21	429.8977273, 281	204, 352, 360, 508
22	628.8977273, 281	204, 551, 360, 707
23	216.7800587, 281	221, 156, 343, 278

Table 6.6: Circle information extracted from *Fsa2*.

Directed line	**Circle**
(217 , 199)	(295 , 204)
(217 , 199)	(508 , 204)
(217 , 199)	(278 , 221)
(430 , 199)	(508 , 204)
(430 , 199)	(707 , 204)
(85 , 283)	(295 , 204)
(85 , 283)	(278 , 221)
(687 , 283)	(707 , 204)
(9 , 299)	(295 , 204)
(9 , 299)	(278 , 221)
(426 , 410)	(508 , 204)
(426 , 410)	(707 , 204)
(213 , 423)	(295 , 204)
(213 , 423)	(508 , 204)

Table 6.7: Arc-node pairing information for *Fsa2*. First set of values is part of the bounding box values of an arc, while the second set is from the paired circle.

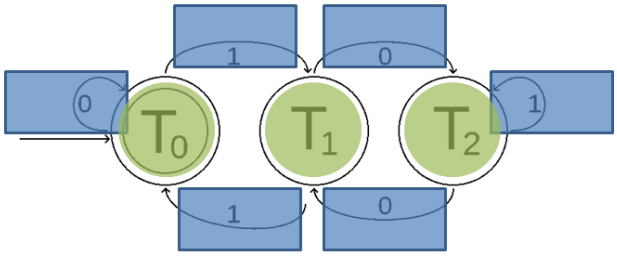


Figure 6.37: The primitives in areas highlighted in green are reduced to states.

a start state. The line is then reduced. In Figure 6.39 the arrow points to an oval shape highlighting the region operated on by the production rule.

Rule 5 searches for labeled arcs (non-terminals obtained from the first rule) in the `ParseForest`. It then converts those to transitions, if they have source and target states. The states are not reduced due to the fact that they are referenced through existential quantifications.

Rule 6 collects multisets for the various states and transitions; effectively the whole diagram has been recognized and the primitives have all been reduced into various non-terminals and finally into the complete diagram. Figure 6.40 shows the various objects affected by this production highlighted by the overlying ellipse.

During parsing, the database is searched several times per production, and lo-

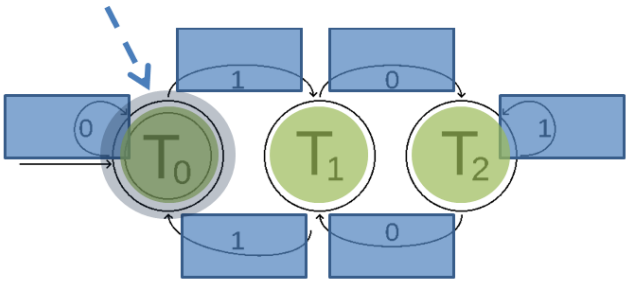


Figure 6.38: Accepting state is highlighted by a dotted arrow. Rule 3 creates a start state from the multiset of circle and state.

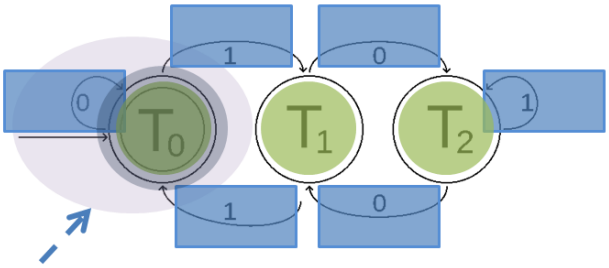


Figure 6.39: The start state region is highlighted by an arrow with dotted line.

cating the particular symbols matching the various constraints in the production rules is not a trivial task. The efficiency of the parsing process can be improved by the use of a database structure which makes the retrieval of these objects faster. In the parser implementation of [73], a *kd-tree* was used.

The figures used for illustrating the parsing process show that the various objects matching the constraints involved in a production are often positioned about the same area in the diagram. This phenomenon can be exploited in the parsing process; instead of naively searching all symbols of the type stated in a production, the parser searches objects around a local region.

This region can be established as a pre-set offset from the primary object involved in the production. For example, if one object matching the set involved in a production rule has been found anchored at a point (x, y), objects positioned at a distance i in the four cardinal directions away from point (x, y) will be the primary focus, and the search scope can be expanded if the object is not found within that primary focus area.

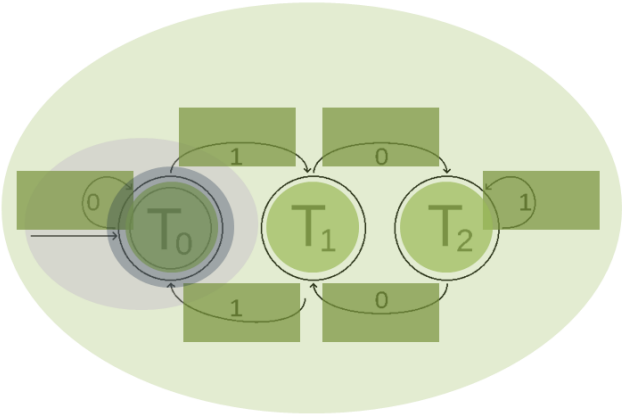


Figure 6.40: All primitives are treated and the diagram completely interpreted.

6.3.1 Summary of results

The first part of this work was to show that the required data for the semantic interpretation of a FSA diagram can be automatically obtained from a raw diagram image, and to show a practical approach to achieving the extraction.

The results from our recognition system shows circles, arcs, arrowheads, and junctions are detected and recognized reasonably well. A summary of the results for the recognition of the FSA diagrams used in this discussion are shown in Tables 6.8, 6.9, and 6.10.

Diagram dimensions(row, column)	(360, 756)
Total number of foreground pixels	4158
Number of detected circles in the image	8
Number of detected arcs in the image	15
Number of detected text characters in the image	2
Number of undetected elements in the image	0
Number of detected node-link connections in the image	30
Total time taken by the system (in seconds)	6.04

Table 6.8: Detection summary for *Fsa1*. This result is after text-graphics separation, and reconstruction.

The key weakness in our proposed system currently lies with the separation of lines from the symbols. Further work needs to be carried out on the task of separating lines and nodes without breaking the node borders, and at the same time ensuring that all node-link junctions are disconnected.

Diagram dimensions(row, column)	(600, 800)
Total number of foreground pixels	3825
Number of detected circles in the image	4
Number of detected arcs in the image	7
Number of detected text characters in the image	12
Number of undetected elements in the image	0
Number of detected node-link connections in the image	14
Total time taken by the system (in seconds)	5.18

Table 6.9: Detection summary for *Fsa2*.

Diagram dimensions(row, column)	(304, 1013)
Total number of foreground pixels	4612
Number of detected circles in the image	6
Number of detected arcs in the image	7
Number of detected text characters in the image	13
Number of undetected elements in the image	1
Number of detected node-link connections in the image	14
Total time taken by the system (in seconds)	5.05

Table 6.10: Detection summary for *Fsa3*.

If these processes are successfully executed, the effectiveness of the recognition system will be closer to that of an application which can be used in everyday analysis of node-link diagrams.

Chapter 7

Conclusion

The task of developing a real-world practical recognition system for the automatic interpretation of Finite State Automata (FSA) diagrams has been examined. This task spans the whole chain of processes necessary to process a raster diagram image, analyze it, and obtain a representation of the contents of the diagram, according to the notation used in encoding the information.

This task is a fairly complex one as other major subtasks are involved. In solving the sub-problems, there are also different techniques to choose from and these need to be integrated. Our approach attempts to pattern the recognition process as a series of operations abstractly modeled after compilers. The aspects involved in the recognition system are grouped into two main phases. One phase deals with the concrete structure of the diagram (lexical aspect) and the other deals with the syntax and semantic aspects of the diagram. Having distinguished two clearly different but interrelated subtasks, the problem of finding a way to solve the tasks, remained. Note that we assumed an input image of reasonable quality from printed pages, and a high-quality scan of the image.

In terms of structure, node-link diagrams (the family of diagrams to which FSA diagrams belong) have relatively simple structures, compared to other technical diagrams. We showed that the lexical extraction process of the constituents of these diagrams can be handled by known image processing and pattern recognition methods. Our own implementation did not include all possibilities in the graphics recognition system, due to time constraints. For example, we did not implement the recognition of intersecting lines, and the node-link separation phase can also be improved. However, we pointed out well-known solutions in such cases.

For interpreting the semantics of the diagram according to the domain, we came to the conclusion that Constraint Multiset Grammars (CMG) were the best option; serving as a suitable alternative to developing an ad hoc grammar. CMGs do not restrict the scope of allowed spatial relations which can be modeled, and neither are limits placed on the attributes of diagram objects (however they must be computable). This is in sharp contrast to other grammars where connection properties

are pre-defined.

CMGs provide a means for describing the syntax of diagram notations. The grammar further provides a means for the analysis of diagram structure from the collection of diagram tokens and their features as extracted from the diagram. The problem of adapting string type grammars to analyzing multi-dimensional representations is eliminated by using the CMG formalism.

Our observation is that CMGs have not been used in offline recognition as much as they have been used in sketch recognition and smart diagram applications. This fact notwithstanding, the confines of static diagram parsing is within the scope of dynamic parsing when it comes to CMGs. As earlier noted in Chapter 5, the parsing features required for online recognition is more than that required by offline recognition.

Finally, we observe that our two phase approach enables a common framework for addressing the diagram recognition problem, as it is straightforward to adapt to different diagram types. Such a framework allows experts to work on different aspects of the problem.

This can lead to more practical and application-friendly deliverables from research work. For instance, our work had to start from the pixels up, because there was no ready made tool which could supply a representation of the information contained in the diagram.

Previous work on diagram recognition for assistive technologies was often hampered by not having an abstract meaning for the symbols. Our research framework removes that problem, as the output of our system enables applications and tools to be built based on the interpretation/meaning of the symbols in the diagram. Hence, it provides the synergy that combines diverse existing techniques and approaches.

Our final conclusion is that a real-world practical automatic recognition system for node-link diagrams is possible, and moreover, we also showed how our solution provides a generalized approach that can be applicable to many other types of diagrams (both at the structural and the semantic levels of recognition).

7.1 Future work

A first obvious route for future work is to implement a full working prototype of our proof of concept system. This would involve, as a first step, the automatic extraction of a diagram from a printed page.

Some limitations earlier mentioned affects the lexical analysis (structural recognition) phase of the system. Therefore our lexical analysis must be improved, particularly as far as intersecting lines and node-link separation is concerned. This would further improve the effectiveness and value of the system since non-planar graphs would be correctly recognized.

An improved CMG parser has to be implemented, possibly with error-correction added to improve the robustness of the recognition and interpretation. In the same scope, the output of the interpretation phase has to be formalized for use in assistive technology tools.

Furthermore, assistive technology tools can be investigated for optimal presentation of node-link diagrams to the blind. In particular, the navigation of large diagrams can be problematic, and little work has been done in this area.

Finally, it would be interesting to consider the robustness of our framework as far as other node-link diagrams are concerned. The importance of carrying out such work lies in the fact that graphs are being used in a large number of fields and for wide range of purposes. For example, other Computer Science diagrams such as entity-relationship diagrams and flowcharts can be considered, or other general diagrams such as abstract railway route diagrams.

List of References

[1] (2014). Enthought Canopy (Version 1.4.1) [Software]. http://www.enthought.com. Accessed December 2014.

[2] (2014). Module: measure - skimage v0.12dev docs [Documentation]. http://scikitimage.org/docs/dev/api/skimage.measure.html. Accessed December 2014.

[3] Ablameyko, S. and Pridmore, T. (2000). *Machine Interpretation of Line Drawing Images: Technical Drawings, Maps and Diagrams*. Springer.

[4] Adam, S., Ogier, J.-M., Cariou, C., Mullot, R., Labiche, J. and Gardes, J. (2000). Symbol and character recognition: application to engineering drawings. *International Journal on Document Analysis and Recognition*, vol. 3, no. 2, pp. 89–101.

[5] Alty, J.L. and Rigas, D.I. (1998). Communicating graphical information to blind users using music: the role of context. In: *Proceedings of the Special Interest Group on Computer-Human Interaction (SIGCHI) Conference on Human Factors in Computing Systems*, pp. 574–581. ACM.

[6] Auer, C., Bachmaier, C., Brandenburg, F., Gleißnerr, A. and Reislhuber, J. (2013). Optical graph recognition. In: Didimo, W. and Patrignani, M. (eds.), *Graph Drawing*, vol. 7704 of *Lecture Notes in Computer Science*, pp. 529–540. Springer. Available at: http://dx.doi.org/10.1007/978-3-642-36763-2_47

[7] Balik, S.P., Mealin, S.P., Stallmann, M.F. and Rodman, R.D. (2013). GSK: universally accessible graph sketching. In: *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, pp. 221–226. ACM.

[8] Beichel, R. (1997). Shape representation and description: contour-based shape representation and description. http://www.engineering.uiowa.edu/~dip/Lecture/Shape2.html/. Accessed February 24, 2014.

[9] Bennett, D. (2002). Effects of navigation and position on task when presenting diagrams to blind people using sound. In: Hegarty, M., Meyer, B. and Narayanan, N. (eds.), *Diagrammatic Representation and Inference*, vol. 2317 of *Lecture Notes in Computer Science*, pp. 161–175. Springer. Available at: http://dx.doi.org/10.1007/3-540-46037-3_19

[10] Bierman, V.J. (1995). *Graphical Scanning of Diagrams–A Spatial Relations Analyzer*. Master's thesis, Leiden University, the Netherlands.

[11] Bland, C.J., Meurer, L.N. and Maldonado, G. (1995). A systematic approach to conducting a non-statistical meta-analysis of research literature. *Academic Medicine*, vol. 70, no. 7, pp. 642–53.

[12] Blenkhorn, P. and Evans, D.G. (1998). Using speech and touch to enable blind people to access schematic diagrams. *Journal of Network and Computer Applications*, vol. 21, no. 1, pp. 17–29.

[13] Bley, H. (1984). Segmentation and preprocessing of electrical schematics using picture graphs. *Computer Vision, Graphics, and Image Processing*, vol. 28, no. 3, pp. 271–288.

[14] Blostein, D. (1996). General diagram-recognition methodologies. In: Kasturi, R. and Tombre, K. (eds.), *Graphics Recognition Methods and Applications*, vol. 1072 of *Lecture Notes in Computer Science*, pp. 106–122. Springer.
Available at: http://dx.doi.org/10.1007/3-540-61226-2_10

[15] Blostein, D. (2000). Defining the syntax and semantics of natural visual languages. In: *Applications of Graph Transformations with Industrial Relevance*, vol. 1779 of *Lecture Notes in Computer Science*, pp. 225–232. Springer.
Available at: http://dx.doi.org/10.1007/3-540-45104-8_16

[16] Blostein, D., Fahmy, H. and Grbavec, A. (1996). Issues in the practical use of graph rewriting. In: Cuny, J., Ehrig, H., Engels, G. and Rozenberg, G. (eds.), *Graph Grammars and Their Application to Computer Science*, vol. 1073 of *Lecture Notes in Computer Science*, pp. 38–55. Springer.
Available at: http://dx.doi.org/10.1007/3-540-61228-9_78

[17] Blostein, D., Lank, E. and Zanibbi, R. (2000). Treatment of diagrams in document image analysis. In: Anderson, M., Cheng, P. and Haarslev, V. (eds.), *Theory and Application of Diagrams*, vol. 1889 of *Lecture Notes in Computer Science*, pp. 330–344. Springer.
Available at: http://dx.doi.org/10.1007/3-540-44590-0_29

[18] Boggess, L. (1999). A fast, inexpensive means of creating tactile "drawings" of graphs and networks for blind students. In: *29th Annual Frontiers in Education Conference, 1999 FIE'99*, vol. 2, pp. 12C5/19–12C5/23. IEEE.

[19] Boshernitsan, M. and Downes, M.S. (2004). Visual programming languages: a survey. Tech. Rep. UCB/CSD-04-1368, Computer Science Division (EECS), University of California, Berkeley, California.
Available at: http://wwww.eecs.berkeley.edu/Pubs/TechRpts/2004/CSD-04-1368.pdf

[20] Bottoni, P., Meyer, B. and Parisi Presicce, F. (2001). Visual multiset rewriting: applications to diagram parsing and reasoning. In: Calude, C., Pàun, G., Rozenberg, G. and Salomaa, A. (eds.), *Multiset Processing*, vol. 2235 of *Lecture Notes in Computer Science*, pp. 45–67. Springer.

[21] Bunke, H. (1982). Attributed programmed graph grammars and their application to schematic diagram interpretation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 6, pp. 574–582.

[22] Burger, W. and Burge, M.J. (2009). *Principles of Digital Image Processing: Core Algorithms*. Springer.

[23] Calder, M., Cohen, R.F., Lanzoni, J. and Xu, Y. (2006). PLUMB: an interface for users who are blind to display, create, and modify graphs. In: *Proceedings of the 8th International ACM Special Interest Group on Accessibility and Computing (SIGACCESS) Conference on Computers and Accessibility*, pp. 263–264. ACM.

[24] Califf, M.E., Goodwin, M.M. and Brownell, J. (2008). Helping him see: guiding a visually impaired student through the computer science curriculum. *ACM Special Interest Group on Computer Science Education (SIGCSE) Bulletin*, vol. 40, no. 1, pp. 444–448.

[25] Chan, C. and Sandler, M. (1992). A complete shape recognition system using the Hough transform and neural network. In: *Pattern Recognition, 1992. Vol. II, Conference B: Proceedings of the 11th International Association for Pattern Recognition (IAPR) International Conference on Pattern Recognition Methodology and Systems*, pp. 21–24. IEEE.

[26] Chang, S.-K. (1987). Visual languages: a tutorial and survey. In: *Selected Contributions on Visualization in Programming. 5th Interdisciplinary Workshop in Informatics and Psychology*, pp. 1–23. Springer.
Available at: http://dl.acm.org/citation.cfm?id=36222.36223

[27] Chhabra, A. (1998). Graphic symbol recognition: an overview. In: Tombre, K. and Chhabra, A. (eds.), *Graphics Recognition Algorithms and Systems*, vol. 1389 of *Lecture Notes in Computer Science*, pp. 68–79. Springer.
Available at: http://dx.doi.org/10.1007/3-540-64381-8_40

[28] Chiousemoglou, M. and Jürgensen, H. (2011). Setting the table for the blind. In: *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*, p. 1. ACM.

[29] Chok, S.S. and Marriott, K. (1995). Automatic construction of user interfaces from Constraint Multiset Grammars. In: *Proceedings of the 11th IEEE International Symposium on Visual Languages*, pp. 242–249. IEEE.

[30] Chok, S.S. and Marriott, K. (1998). Automatic construction of intelligent diagram editors. In: *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, pp. 185–194. ACM.

[31] Chok, S.S., Marriott, K. and Pato, T. (1999). Constraint-based diagram beautification. In: *Proceedings of the 1999 IEEE Symposium on Visual Languages, 1999*, pp. 12–19. IEEE.

[32] Cordella, L. and Vento, M. (2000). Symbol and shape recognition. In: Chhabra, A. and Dori, D. (eds.), *Graphics Recognition Recent Advances*, vol. 1941 of *Lecture Notes in Computer Science*, pp. 167–182. Springer.
Available at: http://dx.doi.org/10.1007/3-540-40953-X_14

[33] Costagliola, G., De Lucia, A., Orefice, S. and Tortora, G. (1997). A framework of syntactic models for the implementation of visual languages. In: *Proceedings of the IEEE Symposium on Visual Languages, 1997*, pp. 58–65. IEEE.

[34] Costagliola, G., Deufemia, V. and Polese, G. (2007). Visual language implementation through standard compiler–compiler techniques. *Journal of Visual Languages and Computing*, vol. 18, no. 2, pp. 165 – 226.

[35] Costagliola, G. and Polese, G. (2000). Extended positional grammars. In: *Proceedings of the IEEE International Symposium on Visual Languages, 2000*, pp. 103–110. IEEE.

[36] Coüasnon, B. (2001). DMOS: a generic document recognition method, application to an automatic generator of musical scores, mathematical formulae and table structures recognition systems. In: *Proceedings of the Sixth International Conference on Document Analysis and Recognition, 2001*, pp. 215–220. IEEE.

[37] Coüasnon, B. (2004). Dealing with noise in DMOS, a generic method for structured document recognition: an example on a complete grammar. In: Llados, J. and Kwon, Y.-B. (eds.), *Graphics Recognition Recent Advances and Perspectives*, vol. 3088 of *Lecture Notes in Computer Science*, pp. 38–49. Springer.
Available at: http://dx.doi.org/10.1007/978-3-540-25977-0_4

[38] Coüasnon, B. (2006). DMOS, a generic document recognition method: application to table structure analysis in a general and in a specific way. *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 8, no. 2-3, pp. 111–122.
Available at: http://dx.doi.org/10.1007/s10032-005-0148-5

[39] Coüasnon, B. and Camillerapp, J. (1994). Using grammars to segment and recognize music scores. In: *Proceedings of the IAPR International Workshop on Document Analysis Systems (DAS-94), October 18-20, Kaiserslautern, Germany*, pp. 15–27. International Association for Pattern Recognition, Kaiserslautern.

[40] Coüasnon, B. and Rétif, B. (1995). Using a grammar for a reliable full score recognition system. In: *Proceedings of the International Computer Music Conference (ICMC), 1995*, pp. 187– 94. Retrieved February 2015.
Available at: http://quod.lib.umich.edu/cgi/p/pod/dod-idx/using-a-grammar-for-a-reliable-full-score-recognition-system.pdf?c=icmc;idno=bbp2372.1995.057

[41] Crimi, C., Guercio, A., Nota, G., Pacini, G., Tortora, G. and Tucci, M. (1991). Relation grammars and their application to multi-dimensional languages. *Journal of Visual Languages and Computing*, vol. 2, no. 4, pp. 333 – 346.
Available at: http://www.sciencedirect.com/science/article/pii/S1045926X05800035

[42] Davies, E.R. (2004). *Machine Vision: Theory, Algorithms, Practicalities*. Elsevier.

[43] Dibeklioglu, H. (2006). A sketch recognizer for graphs. Tech. Rep., Department of Computer Engineering, Yeditepe University, Istanbul, Turkey.
Available at: https://staff.fnwi.uva.nl/h.dibeklioglu/documents/Dibeklioglu2006EngineeringProjectReport.pdf

[44] Doermann, D.S. (1998). An introduction to vectorization and segmentation. In: *Selected Papers from the Second International Workshop on Graphics Recognition, Algorithms and Systems*, GREC '97, pp. 1–8. Springer.
Available at: http://dl.acm.org/citation.cfm?id=645437.652755

[45] Dori, D. (2000). Syntactic and semantic graphics recognition: the role of the object-process methodology. In: Chhabra, A. and Dori, D. (eds.), *Graphics Recognition Recent Advances*, vol. 1941 of *Lecture Notes in Computer Science*, pp. 277–287. Springer.
Available at: http://dx.doi.org/10.1007/3-540-40953-X_24

[46] Dori, D. and Wenyin, L. (1999). Automated CAD conversion with the machine drawing understanding system: concepts, algorithms, and performance. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 29, no. 4, pp. 411–416.

[47] Dosch, P., Tombre, K., Ah-Soon, C. and Masini, G. (2000). A complete system for the analysis of architectural drawings. *International Journal on Document Analysis and Recognition*, vol. 3, no. 2, pp. 102–116.

[48] Edwards, B. and Chandran, V. (2000). Machine recognition of hand-drawn circuit diagrams. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2000, ICASSP'00*, vol. 6, pp. 3618–3621. IEEE.

[49] Ejiri, M., Kakumoto, S., Miyatake, T., Shimada, S. and Iwamura, K. (1990). Automatic recognition of engineering drawings and maps. In: Kasturi, R. (ed.), *Image analysis applications*, vol. 24, pp. 73–126. CRC Press.

[50] Engelhardt, Y. (2002). *The Language of Graphics: A Framework for the Analysis of Syntax and Meaning in Maps, Charts and Diagrams*. Ph.D. thesis, Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands.

[51] Eppler, M.J. (2006). A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing. *Information Visualization*, vol. 5, no. 3, pp. 202–210.

[52] Fahmy, H. and Blostein, D. (1992). A survey of graph grammars: theory and applications. In: *Proceedings of the 11th IAPR International Conference on Pattern Recognition, 1992. Vol.II, Conference B: Pattern Recognition Methodology and Systems*, pp. 294–298. IEEE.

[53] Farahmand, A., Sarrafzadeh, A. and Shanbehzadeh, J. (2013). Document image noises and removal methods. In: *Proceedings of the International Multiconference of Engineers and Computer Scientists*, vol. 1, pp. 13–15. Newswood Limited.

[54] Felzenszwalb, P.F. (2003). *Representation and Detection of Shapes in Images*. Ph.D. thesis, Massachusetts Institute of Technology.
Available at: http://cs.brown.edu/~pff/papers/pff.pdf

[55] Ferrucci, F., Pacini, G., Satta, G., Sessa, M., Tortora, G., Tucci, M. and Vitiello, G. (1996). Symbol-relation grammars: a formalism for graphical languages. *Information and Computation*, vol. 131, no. 1, pp. 1 – 46.
Available at: http://www.sciencedirect.com/science/article/pii/S0890540196900905

[56] Fisher, R. (2005). *Dictionary of Computer Vision and Image Processing*. Wiley.

[57] Fitch, W. and Friederici, A. (2012). Artificial grammar learning meets formal language theory: an overview. *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 367, no. 1598, pp. 1933–1955.
Available at: http://groups.lis.illinois.edu/amag/langev/paper/fitch2012artificialgsc.html

[58] Fletcher, L.A. and Kasturi, R. (1988). A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 6, pp. 910–918.

[59] Födisch, M., Crombie, D. and Ioannidis, G. (2002). TeDUB: providing access to technical drawings for print impaired people. In: *Proceedings of Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment: Accessibility, Mobility and Social Integration*.
Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.9102&rep=rep1&type=pdf

[60] Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *Institute of Radio Engineers (IRE) Transactions on Electronic Computers*, vol. EC-10, no. 2, pp. 260–268.

[61] Fu, K. (1979). Recent advances in syntactic pattern recognition. In: Haken, H. (ed.), *Pattern Formation by Dynamic Systems and Pattern Recognition*, vol. 5 of *Springer Series in Synergetics*, pp. 176–185. Springer.
Available at: http://dx.doi.org/10.1007/978-3-642-67480-8_17

[62] Fu, K. (1982). *Syntactic Pattern Recognition and Applications*. Prentice-Hall Advanced Reference Series: Computer Science. Prentice-Hall.
Available at: http://books.google.co.za/books?id=EigZAQAAIAAJ

[63] Fujisawa, H. (2008). Forty years of research in character and document recognition – an industrial perspective. *Pattern Recognition*, vol. 41, no. 8, pp. 2435–2446.

[64] Futrelle, R.P. (1990). Strategies for diagram understanding: object/spatial data structures, animate vision, and generalized equivalence. In: *Proceedings of the 10th International Conference on Pattern Recognition (ICPR)*, pp. 403–408. IEEE.

[65] Futrelle, R.P., Kakadiaris, I.A., Alexander, J., Carriero, C.M., Nikolakis, N. and Futrelle, J.M. (1992). Understanding diagrams in technical documents. *Computer*, vol. 25, no. 7, pp. 75–78.

[66] Futrelle, R.P. and Nikolakis, N. (1995). Efficient analysis of complex diagrams using constraint-based parsing. In: *Proceedings of the Third International Conference on Document Analysis and Recognition, 1995*, vol. 2, pp. 782–790. IEEE.

[67] Goldwasser, S.M. (2003). Everton gas tube sign power supply schematic. http://www.repairfaq.org/sam/etrn3210.gif/. Accessed February 24, 2014.

[68] Golin, E.J. (1991). A method for the specification and parsing of visual languages. Tech. Rep. CS-90-19, Department of Computer Science, Brown University, Providence, Rhode Island, United States.
Available at: ftp://ftp.cs.brown.edu/pub/techreports/90/cs90-19.pdf

[69] Golin, E.J. and Reiss, S.P. (1990). The specification of visual language syntax. *Journal of Visual Languages and Computing*, vol. 1, no. 2, pp. 141 – 157.
Available at: http://www.sciencedirect.com/science/article/pii/S1045926X05800138

[70] Gorman, L. (1995). *Document Image Analysis*. IEEE Computer Society Press, Los Alamitos, California.

[71] Hamada, A.H. (1993). A new system for the analysis of schematic diagrams. In: *Proceedings of the Second International Conference on Document Analysis and Recognition, 1993*, pp. 369–372. IEEE.

[72] Haralick, R., Sternberg, S.R. and Zhuang, X. (1987). Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 532–550.

[73] Helm, R., Marruitt, K. and Odersky, M. (1991). Building visual language parsers. In: *Proceedings of the Special Interest Group on Computer-Human Interaction (SIGCHI) Conference on Human Factors in Computing Systems*, pp. 105–112. ACM.

[74] Henderson, T.C. and Samal, A. (1986). Shape grammar compilers. *Pattern Recognition*, vol. 19, no. 4, pp. 279 – 288.
Available at: http://www.sciencedirect.com/science/article/pii/0031320386900531

[75] Hirayama, M.J. and Kimura, Y. (2011). Beading design diagram reading system for visually impaired persons. In: *17th Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV), 2011*, pp. 1–4. IEEE.

[76] Horstmann, M., Hagen, C., King, A., Dijkstra, S., Crombie, D., Evans, D.G., Ioannidis, G.T., Blenkhorn, P., Herzog, O. and Schlieder, C. (2004). TeDUB: automatic interpretation and presentation of technical diagrams for blind people. In: Hersh, M. (ed.), *Proceedings of Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment - CVHI'2004, EURO-ASSIST-VHI-2: Accessibility, Mobility and Social Integration, (Granada, Spain, 2004)*.
Available at: http://www-agki.tzi.de/grp/ag-ki/download/2004/horstmannetal_cvhi2004.pdf

[77] Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *Institute of Radio Engineers (IRE) Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187.

[78] Huang, W., Tan, C. and Leow, W. (2004). Model-based chart image recognition. In: Llados, J. and Kwon, Y.-B. (eds.), *Graphics Recognition Recent Advances and Perspectives*, vol. 3088 of *Lecture Notes in Computer Science*, pp. 87–99. Springer.
Available at: http://dx.doi.org/10.1007/978-3-540-25977-0_8

[79] Iizuka, K., Tanaka, J. and Shizuki, B. (2001). Describing a drawing editor by using Constraint Multiset Grammars. In: *Proceedings of the Sixth International Symposium on the Future of Software Technology*.
Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.8831&rep=rep1&type=pdf

[80] Joung, S. and Tanaka, J. (2000). Generating a visual system with soft layout constraints. In: *Proceedings of the International Conference on Information (Information'2000)*, pp. 138–145. Accessed April, 2, 2015.
Available at: http://www.iplab.is.tsukuba.ac.jp/paper/international/joung-i2000.ps.gz

[81] Kaneko, T. (1992). Line structure extraction from line-drawing images. *Pattern Recognition*, vol. 25, no. 9, pp. 963–973.

[82] Kanungo, T., Haralick, R. and Dori, D. (1995). Understanding engineering drawings: a survey. In: *Proceedings of First IAPR Workshop on Graphics Recognition*, pp. 217–228. University Park, PA.
Available at: http://www.kanungo.com/pubs/iwgr95-survey.ps

[83] Karasneh, B. and Chaudron, M. (2013). Extracting UML models from images. In: *5th International Conference on Computer Science and Information Technology (CSIT), 2013*, pp. 169–178. IEEE.

[84] Kasturi, R., Bow, S.T., El-Masri, W., Shah, J., Gattiker, J.R. and Mokate, U.B. (1990). A system for interpretation of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 978–992.

[85] Kasturi, R., Bow, S.-T., Gattiker, J., Shah, J., El-Masri, W., Mokate, U. and Honnenahalli, S. (1988). A system for recognition and description of graphics. In: *9th International Conference on Pattern Recognition, 1988*, pp. 255–259. IEEE.

[86] Kasturi, R., O'Gorman, L. and Govindaraju, V. (2002). Document image analysis: a primer. *Sadhana*, vol. 27, no. 1, pp. 3–22.
Available at: http://dx.doi.org/10.1007/BF02703309

[87] Kawai, Y., Ohnishi, N. and Sugie, N. (1990). A support system for the blind to recognize a diagram. *Systems and Computers in Japan*, vol. 21, no. 7, pp. 75–85.

[88] Kennel, A.R. (1996). Audiograf: a diagram-reader for the blind. In: *Proceedings of the Second Annual ACM Conference on Assistive Technologies*, pp. 51–56. ACM.

[89] Kim, S., Suh, J. and Kim, J. (1993). Recognition of logic diagrams by identifying loops and rectilinear polylines. In: *Proceedings of the Second International Conference on Document Analysis and Recognition, 1993*, pp. 349–352. IEEE.

[90] King, A., Blenkhorn, P., Crombie, D., Dijkstra, S., Evans, G. and Wood, J. (2004). Presenting UML software engineering diagrams to blind people. *Computers Helping People with Special Needs*, pp. 522–529.

[91] Lai, C. and Kasturi, R. (1991). Detection of dashed lines in engineering drawings and maps. In: *Proceedings of the First International Conference on Document Analysis and Recognition, Saint-Malo (France)*, vol. 2, pp. 507–515.
Available at: http://www.iapr-tc11.org/archive/icdar1991_proc/proceedings/507.pdf

[92] Lakin, F. (1986). Spatial parsing for visual languages. In: Chang, S.-K., Ichikawa, T. and Ligomenides, P. (eds.), *Visual Languages*, Management and Information Systems, pp. 35–85. Springer.
Available at: http://dx.doi.org/10.1007/978-1-4613-1805-7_3

[93] Lam, L., Lee, S.-W. and Suen, C.Y. (1992). Thinning methodologies - a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 9, pp. 869–885.

[94] Lamiroy, B. and Ogier, J.-M. (2014). Analysis and interpretation of graphical documents. In: Doermann, D. and Tombre, K. (eds.), *Handbook of Document Image Processing and Recognition*, pp. 553–590. Springer.
Available at: http://dx.doi.org/10.1007/978-0-85729-859-1_19

[95] Levner, I. (2002). Shape detection, analysis and recognition. http://webdocs.cs.ualberta.ca/~ilya/courses/c615-PerceptualSystems/c615_Final_paper.pdf.gz. Accessed January 31, 2015.

[96] Li, L. and Tam, C.L. (2008). A graphics image processing system. In: *The Eighth IAPR International Workshop on Document Analysis Systems, 2008 (DAS '08)*, pp. 455–462. IEEE.

[97] Liao, S.X. and Pawlak, M. (1996). On image analysis by moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 3, pp. 254–266.

[98] Lladós, J., Valveny, E., Sánchez, G. and Martí, E. (2002). Symbol recognition: current advances and perspectives. In: Blostein, D. and Kwon, Y.-B. (eds.), *Graphics Recognition Algorithms and Applications*, vol. 2390 of *Lecture Notes in Computer Science*, pp. 104–128. Springer.
Available at: http://dx.doi.org/10.1007/3-540-45868-9_9

[99] Loncaric, S. (1998). A survey of shape analysis techniques. *Pattern Recognition*, vol. 31, no. 8, pp. 983–1001.

[100] Lu, T., Yang, H., Yang, R. and Cai, S. (2007). Automatic analysis and integration of architectural drawings. *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 9, no. 1, pp. 31–47.

[101] Lu, Z. (1998). Detection of text regions from digital engineering drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 4, pp. 431–439.

[102] Macé, S. and Anquetil, E. (2007). Design of a pen-based electric diagram editor based on Context-Driven Constraint Multiset Grammars. In: Jacko, J. (ed.), *Human-Computer Interaction Interaction Platforms and Techniques*, vol. 4551 of *Lecture Notes in Computer Science*, pp. 418–428. Springer.
Available at: http://dx.doi.org/10.1007/978-3-540-73107-8_47

[103] Macé, S. and Anquetil, E. (2009). Eager interpretation of on-line hand-drawn structured documents: the DALI methodology. *Pattern Recognition*, vol. 42, no. 12, pp. 3202 – 3214. New Frontiers in Handwriting Recognition.
Available at: http://www.sciencedirect.com/science/article/pii/S0031320308004482

[104] Marques, O. (2011). *Practical Image and Video Processing Using MATLAB*. Wiley-IEEE Press, Hoboken NJ.

[105] Marriott, K. (1994). Constraint Multiset Grammars. In: *Proceedings of the IEEE Symposium on Visual Languages*, pp. 118–125. IEEE.

[106] Marriott, K. and Meyer, B. (1997). On the classification of visual languages by grammar hierarchies. *Journal of Visual Languages and Computing*, vol. 8, no. 4, pp. 375 – 402.

[107] Marriott, K. and Meyer, B. (1998). The CCMG visual language hierarchy. In: Marriott, K. and Meyer, B. (eds.), *Visual Language Theory*, pp. 129–169. Springer.

[108] Marriott, K. and Meyer, B. (eds.) (1998). *Visual Language Theory*. Springer.

[109] Marriott, K., Meyer, B. and Wittenburg, K. (1998). A survey of visual language specification and recognition. In: Marriott, K. and Meyer, B. (eds.), *Visual Language Theory*, pp. 5–85. Springer.
Available at: http://dx.doi.org/10.1007/978-1-4612-1676-6_2

[110] Marshall, S. (1989). Review of shape coding techniques. *Image and Vision Computing*, vol. 7, no. 4, pp. 281–294.

[111] McDaniel, J.R. and Balmuth, J.R. (1992). Kekule: OCR – optical chemical (structure) recognition. *Journal of Chemical Information and Computer Sciences*, vol. 32, no. 4, pp. 373–378.

[112] Meyer, B. (2000). A constraint-based framework for diagrammatic reasoning. *Applied Artificial Intelligence*, vol. 14, no. 4, pp. 327–344.

[113] Miller, W.F. and Shaw, A.C. (1968). Linguistic methods in picture processing: a survey. In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pp. 279–290. ACM.
Available at: http://doi.acm.org/10.1145/1476589.1476630

[114] Modayur, B.R., Ramesh, V., Haralick, R.M. and Shapiro, L.G. (1993). MUSER: A prototype musical score recognition system using mathematical morphology. *Machine Vision and Applications*, vol. 6, no. 2-3, pp. 140–150.

[115] Mogensen, T.Æ. (2011). *Introduction to Compiler Design*. Springer.

[116] Nagasamy, V. and Langrana, N.A. (1990). Engineering drawing processing and vectorization system. *Computer Vision, Graphics, and Image Processing*, vol. 49, no. 3, pp. 379–397.

[117] Nagl, M. (1979). A tutorial and bibliographical survey on graph grammars. In: *Graph-Grammars and Their Application to Computer Science and Biology*, pp. 70–126. Springer.

[118] Narasimhan, R. (1966). Syntax-directed interpretation of classes of pictures. *Communications of the ACM*, vol. 9, no. 3, pp. 166–173.
Available at: http://doi.acm.org/10.1145/365230.365258

[119] Nisan, N. and Schocken, S. (2008). *The Elements of Computing Systems: Building a Modern Computer from First Principles*. MIT Press.

[120] Nixon, M. and Aguado, A.S. (2008). *Feature Extraction & Image Processing*. Academic Press.

[121] O'Gorman, L. (1996). Basic techniques and symbol-level recognition: an overview. In: Kasturi, R. and Tombre, K. (eds.), *Graphics Recognition Methods and Applications*, vol. 1072 of *Lecture Notes in Computer Science*, pp. 1–12. Springer.
Available at: http://dx.doi.org/10.1007/3-540-61226-2_1

[122] O'Gorman, L., Sammon, M.J. and Seul, M. (2008). *Practical Algorithms for Image Analysis with CD-ROM*. Cambridge University Press, Cambridge.

[123] Pavlidis, T. (1980). Algorithms for shape analysis of contours and waveforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 4, pp. 301–312.

[124] Peura, M. and Iivarinen, J. (1997). Efficiency of simple shape descriptors. In: Arcelli, C., Cordella, L.P. and di Baja, G.S. (eds.), *Advances in Visual Form Analysis*, pp. 443–451. World Scientific, Singapore.

[125] Prerau, D. (1975). DO-RE-MI: A program that recognizes music notation. *Computers and the Humanities*, vol. 9, no. 1, pp. 25–29.
Available at: http://dx.doi.org/10.1007/BF02404318

[126] Ramel, J.-Y. and Vincent, N. (2004). Strategy for line drawing understanding. In: *Graphics Recognition Recent Advances and Perspectives*, vol. 3088 of *Lecture Notes in Computer Science*, pp. 1–12. Springer.
Available at: http://dx.doi.org/10.1007/978-3-540-25977-0_1

[127] Ramloll, R., Yu, W., Brewster, S., Riedel, B., Burton, M. and Dimigen, G. (2000). Constructing sonified haptic line graphs for the blind student: first steps. In: *Proceedings of the Fourth International ACM Conference on Assistive Technologies*, pp. 17–25. ACM.

[128] Rekers, J. and Schurr, A. (1996). A graph based framework for the implementation of visual environments. In: *Proceedings of the IEEE Symposium on Visual Languages, 1996*, pp. 148–155. IEEE.

[129] Rekers, J. and Schürr, A. (1997). Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages and Computing*, vol. 8, no. 1, pp. 27–55.

[130] Rigaud, C., Karatzas, D., Van De Weijer, J., Burie, J.-C. and Ogier, J.-M. (2013). Automatic text localisation in scanned comic books. In: *Proceedings of the 9th International Conference on Computer Vision Theory and Applications (VISAPP)*, vol. 1, pp. 814–819. Barcelona, Spain.
Available at: https://hal.archives-ouvertes.fr/hal-00841492

[131] Roy, P., Vazquez, E., Lladós, J., Baldrich, R. and Pal, U. (2008). A system to segment text and symbols from color maps. In: Liu, W., Lladós, J. and Ogier, J.-M. (eds.), *Graphics Recognition Recent Advances and New Opportunities*, vol. 5046 of *Lecture Notes in Computer Science*, pp. 245–256. Springer.
Available at: http://dx.doi.org/10.1007/978-3-540-88188-9_23

[132] Rusinol, M., de las Heras, L.-P., Mas, J., Terrades, O.R., Karatzas, D., Dutta, A., Sánchez, G. and Lladós, J. (2012). CVC-UAB's participation in the flowchart recognition task of CLEF-IP 2012. In: *CLEF (Online Working Notes/Labs/Workshop)*.
Available at: http://ceur-ws.org/Vol-1178/CLEF2012wn-CLEFIP-RusinolEt2012.pdf

[133] Samet, R. and Hancer, E. (2012). A new approach to the reconstruction of contour lines extracted from topographic maps. *Journal of Visual Communication and Image Representation*, vol. 23, no. 4, pp. 642–647.

[134] Selker, T. and Koved, L. (1988). Elements of visual language. In: *IEEE Workshop on Visual Languages, 1988*, pp. 38–44. IEEE.

[135] Shapiro, L. and Stockman, G. (2001). *Computer Vision*. Prentice Hall.

[136] Shaw, A.C. (1969). A formal picture description scheme as a basis for picture processing systems. *Information and Control*, vol. 14, no. 1, pp. 9–52.

[137] Shizuki, B., Iizuka, K. and Tanaka, J. (2004). Framework for interpreting handwritten strokes using grammars. In: Masoodian, M., Jones, S. and Rogers, B. (eds.), *Computer Human Interaction*, vol. 3101 of *Lecture Notes in Computer Science*, pp. 409–419. Springer.
Available at: http://dx.doi.org/10.1007/978-3-540-27795-8_41

[138] Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology.

[139] Smith, R. (2007). An overview of the Tesseract OCR engine. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR)*, vol. 7, pp. 629–633. IEEE.

[140] Sonka, M. (2008). *Image Processing, Analysis, and Machine Vision*. Thompson Learning, Toronto.

[141] Tim, B. and Jack, M. (2014). Formal languages, Computer Science Field Guide. http://www.cosc.canterbury.ac.nz/csfieldguide/dev/dev/index.html. Accessed December 23, 2014.

[142] Tombre, K. (1996). Structural and syntactic methods in line drawing analysis: to which extent do they work? In: Perner, P., Wang, P. and Rosenfeld, A. (eds.), *Advances in Structural and Syntactical Pattern Recognition*, vol. 1121 of *Lecture Notes in Computer Science*, pp. 310–321. Springer.
Available at: http://dx.doi.org/10.1007/3-540-61577-6_32

[143] Tombre, K., Ah-Soon, C., Dosch, P., Habed, A. and Masini, G. (1998). Stable, robust and off-the-shelf methods for graphics recognition. In: *Proceedings of the Fourteenth International Conference on Pattern Recognition, 1998*, vol. 1, pp. 406–408. IEEE.

[144] Tombre, K. and Tabbone, S. (2000). Vectorization in graphics recognition: to thin or not to thin. In: *Proceedings of the 15th International Conference on Pattern Recognition, 2000*, vol. 2, pp. 91–96. IEEE.

[145] Tombre, K., Tabbone, S. and Dosch, P. (2006). Musings on symbol recognition. In: Liu, W. and Llados, J. (eds.), *Graphics Recognition Ten Years Review and Future Perspectives*, vol. 3926 of *Lecture Notes in Computer Science*, pp. 23–34. Springer.
Available at: http://dx.doi.org/10.1007/11767978_3

[146] Tombre, K., Tabbone, S., Pélissier, L., Lamiroy, B. and Dosch, P. (2002). Text/graphics separation revisited. In: Lopresti, D., Hu, J. and Kashi, R. (eds.), *Document Analysis Systems V*, vol. 2423 of *Lecture Notes in Computer Science*, pp. 200–211. Springer.

[147] Tversky, B. (2001). Spatial schemas in depictions. In: Gattis, M. (ed.), *Spatial Schemas and Abstract Thought*, pp. 79–112. MIT Press / Bradford Books, Cambridge, MA.

[148] Valveny, E., Dosch, P., Winstanley, A., Zhou, Y., Yang, S., Yan, L., Wenyin, L., Elliman, D., Delalandre, M., Trupin, E. *et al.* (2007). A general framework for the evaluation of symbol recognition methods. *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 9, no. 1, pp. 59–74.

[149] Van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E. and Yu, T. (2014). scikit-image: image processing in Python. *PeerJ*, vol. 2, p. e453.
Available at: http://dx.doi.org/10.7717/peerj.453

[150] Vaxivière, P. and Tombre, K. (1992). Celesstin: CAD conversion of mechanical drawings. *Computer*, vol. 25, no. 7, pp. 46–54.

[151] Way, T.P. (1996). *Automatic Generation of Tactile Graphics.* Master's thesis, University of Delaware, Delaware.
Available at: http://www.csc.villanova.edu/~tway/publications/waymsthesis.pdf

[152] Wittenburg, K. (1998). Visual language parsing: if I had a hammer... In: Bunt, H., Beun, R.-J. and Borghuis, T. (eds.), *Multimodal Human-Computer Communication*, vol. 1374 of *Lecture Notes in Computer Science*, pp. 231–249. Springer.
Available at: http://dx.doi.org/10.1007/BFb0052321

[153] Wu, R.-Q., Cheng, X.-R. and Yang, C.-J. (2009). Extracting contour lines from topographic maps based on cartography and graphics knowledge. *Journal of Computer Science and Technology*, vol. 9, pp. 58–64.

[154] Yang Mingqiang, K.K. and Joseph, R. (2008). A survey of shape feature extraction techniques. In: *Pattern Recognition Techniques, Technology and Applications*, chap. 3. InTech, Vienna, Austria.

[155] Yoneda, N., Kise, K., Takamatsu, S. and Fukunaga, K. (1994). A method of understanding conceptual diagrams. In: *IAPR Workshop on Machine Vision Applications, MVA'94*, pp. 334–337.

[156] Yu, W., Ramloll, R. and Brewster, S. (2001). Haptic graphs for blind computer users. In: Brewster, S. and Murray-Smith, R. (eds.), *Haptic Human-Computer Interaction*, vol. 2058 of *Lecture Notes in Computer Science*, pp. 41–51. Springer.
Available at: http://dx.doi.org/10.1007/3-540-44589-7_5

[157] Yu, Y., Samal, A. and Seth, S.C. (1997). A system for recognizing a large class of engineering drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 8, pp. 868–890.

[158] Zapirain, B.G., Zorrilla, A.M., Oleagordia, I.R. and Muro, A. (2010). Accessible schematics content descriptors using image processing techniques for blind students learning. In: *5th International Symposium on I/V Communications and Mobile Network (ISVC), 2010*, pp. 1–4. IEEE.

[159] Zhang, D. and Lu, G. (2004). Review of shape representation and description techniques. *Pattern Recognition*, vol. 37, no. 1, pp. 1–19.

[160] Zhang, D., Zhang, K. and Cao, J. (2001). A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Journal*, vol. 44, no. 3, pp. 186–200.

[161] Zhang, T. and Suen, C.Y. (1984). A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, vol. 27, no. 3, pp. 236–239.

[162] Zhou, Y. and Tan, C.L. (2001). Chart analysis and recognition in document images. In: *Proceedings of the Sixth International Conference on Document Analysis and Recognition, 2001*, pp. 1055–1058. IEEE.