Recognition of Logic Diagrams by Identifying Loops and Rectilinear Polylines

S. H. Kim , J. W. Suh and J. H. Kim
Dept of Computer Science and Center for AI Research
Korea Advanced Institute of Science & Technology
Daejon 305-701, Korea

Abstract

Proposed is a system that recognizes logic symbols and their inter–connections on logic diagrams. Input diagram, digitized by scanner, is converted into a set of line segments through a sequence of picture processing operations. Then symbols and connections are extracted by identifying loops and rectilinear polylines utilizing model–base in which symbols are graphically described.

Experiment with a number of logic diagrams shows that the system correctly recognizes more than 96 % of logic symbols and connections on A4-size diagram with an average complexity within 15 seconds on a workstation.

1 Introduction

A system for the recognition of schematic diagrams can be evaluated by the three criteria of recognition accuracy, flexibility, and efficiency. Most systems reported in the literature [1, 3, 6] imposed severe restrictions on the input diagrams to improve the recognition accuracy, and are very sensitive to the change of domain knowledge while taking little care to the efficiency. Such restrictions degrade the applicability of those systems. Futhermore modifying of domain knowledge may require rewriting the recognition program. Even worse, they are quite inefficient for pratical diagrams with low quality images of large size.

In this paper we propose an automatic system which recognizes logic diagrams by identifying logic symbols and their interconnections. A degree of recognition accuracy has been reached through the identification of loops and rectilinear polylines. The frame-based description of symbol models makes our system highly flexible. A significant improvement of computational efficiency has been obtained by the use of contour-based picture processing operations. Experiment with a number of logic diagrams shows the superiority of our system.

2 Preprocessing

The preprocessing module consists of a sequence of picture processing operations: separation of text and graphics, extraction of line structures, and line segmentation. Digitized logic diagram is converted into a set of line segments for representing the structural information of lines and regions. Before all the operations, a chain is generated for every closed contour, the outermost layer of the object in the image. These chains are referred by individual operations to reduce the amount of time for locating object pixels and computing shape features of each object.

The preprocessing module runs fast due to elimination of the time consuming search on the entire image, except for the initial contour tracing, and having constituent operations use two bits per pixel for storage.

2.1 Separation of text and graphics

We have devised a simple algorithm for the separation of text from graphics in digitized line drawing images based on the shape features of individual objects and some thresholds. The shape features, — such as width, height, elongatedness, and complexity — are computed from the object boundaries and the variances of the projections along the two principal axes of the object (see Figure 1).

Large-sized or elongated objects are regarded as graphics. We can ever separate text strings whose character components are attached to each other by considering the elongation of the objects. Objects which correspond to text have relatively large values of complexity while objects corresponding to a part of graphics have smaller values.

Most texts of various languages, fonts, and orientations are successively separated by the classifier if they

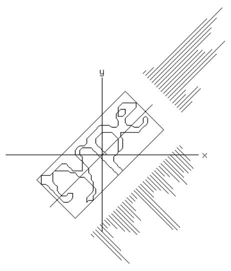


Figure 1: Bounding box and projection of an object

do not overlap with the graphics. The separated textual data is supposed to be processed by a character recognition system.

2.2 Extraction of line structures

To convert the separated graphical objects into a form that is suitable for further processing, those objects are reduced into single-pixel wide *line structures*. Thin lines should be represented by their medial lines or *skeletons* while thick regions are represented by their boundaries or *contours*.

We have utilized the contour generation method in Kwok's thinning algorithm [5]. Given an object contour, the new contour which will be exposed to the background when the current contour is removed is constructed. A set of case–by–case rules are used for the generation.

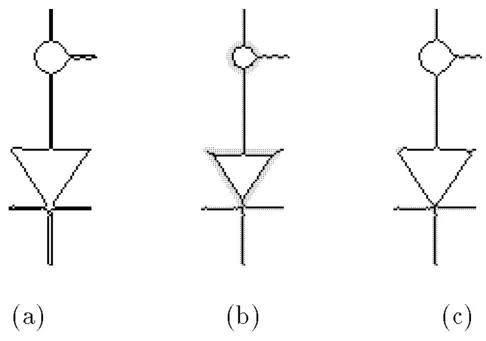


Figure 2: (a) initial contour (b) contracted contour (c) line skeletons and region boundaries

The contour generation in itself *contracts* an object into the inside of the object along its boundaries. Since it distinguishes the object interior from the background with the chain direction of the boundaries, we can adapt the method to *expand* the object toward the background by reversing the direction of the chain.

The proposed algorithm utilizes the fact that a few iterations of contour generation are sufficient to obtain the skeletons of thin lines. Thick regions which have been contracted by the iterations can be restored by the expansion. Figure 2 illustrates the algorithm in operation. When no region is found in the input image, the algorithm just acts as Kwok's thinning algorithm.

2.3 Line segmentation

Because of pool quality of input drawings, there may exist some errors in the line structures: noisy *spurs*, small *gaps*, and *deformations* around junctions. Noisy spurs are simply removed and small gaps are filled by examining the neighborhood area of each end-point. Deformed junctions are corrected by deleting and modifying relevant edges.

After correcting errors, each edge is approximated with lines by the use of the iterative end-point fit algorithm [2]. The resulting lines are represented by a graph, where vertex corresponds to an end of line and edge represents a line segment.

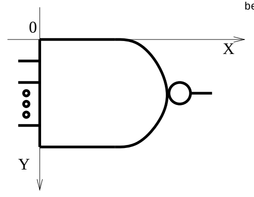
3 Symbol recognition

The symbol recognition module extracts logic symbols and connections from the preprocessed logic diagrams. Among various logic symbols and connections, we confine our attention to *loop-symbols* and *rectilinear connections*. Four orientations (left, right, up, and down) are permitted for a symbol, but the size and position are free. A set of input/output terminals is associated with each symbol. The connection is neither self-crossed nor a closed polygon. Two connections, if they meet, cross each other orthogonally.

3.1 Symbol models

The entire recognition process is guided by model–base which consists of a set of symbol models, each of which is described graphically with a frame-like template. With a such scheme for symbol description, it is easy to create, delete and modify symbols. Figure 3 shows an example.

Here we define some terms. A *characteristic loop* of a symbol model is the largest primitive loop in the symbol. A *symbol class* is a set of symbol models having the same characteristic loop. A *characteristic window* is the bounding rectangle of the image of the union of all the symbols in that class, where all the characteristic loops coincide. In our implementation, 17 loop-symbols are grouped into 6 classes.



```
begin {symbol}
      name   (Nand-Gate)
      width  (132)
      height (90)
      begin {graphical object}
            line (0,0) (67,0)
            line (0,0) (0,90)
            line (0,90) (67,90)
            e-arc (67,45) (45) (45) (-90) (180)
            circle (122,45) (10) (empty)
      end {graphical object}
      begin {input terminal}
            on (line (0,0) (0,90)) (left)
      end {input terminal}
      begin {output terminal}
            at (132,0) (right)
      end {output terminal}

end {symbol}
```

Figure 3: Graphical description of a predefined logic symbol

3.2 Recognition algorithms

Given a preprocessed logic diagram, we first find the primitive loops whose shapes coinside a characteristic loop of a symbol model. We conclude two loops coincide if they differ only by a combination of translation, scaling, and change in size. Starting from a matched primitive loop, loop-symbol is isolated by excluding the connections around the loop. The isolated symbol is then tested whether its shape coincide to one of the symbol models belonging to the symbol class associated with the matched characteristic loop.

To test shape identity, we adopt the features which are invariant under the affine transformations: Fourier descriptors [7] for loops and the moment invariants [4] for symbols, respectively.

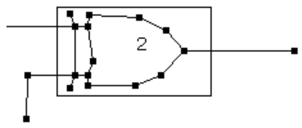


Figure 4: A matched loop and related window

Loop matching: Given a graph of line segments which represents the preprocessed logic diagram, a primitive loop is detected , and its Fourier descriptor , a ten-tuple vector is computed. The loop is identified as one of the characteristic loops that has the minimum Manhattan distance from the Fourier descriptor vector, which is less than a preset threshold.

When a primitive loop γ matches with a characteristic loop ξ, a window W_γ for γ is obtained by transforming the characteristic window W_ξ of ξ according to the pose of γ. Figure 4 shows an example W_γ placed on the graph. There are 15 line segments which are totally enclosed by W_γ and three lines crossing the window.

Extraction of connections: The window W_γ has been determined to completely enclose any symbol containing the loop γ. We can regard each line crossing the window as a part of connection due to the assumption of rectilinearity of connections. Starting from such a line segment l_i, we extract a rectilinear polyline from the graph by merging adjacent lines which meet collinearly or orthogonally with l_i. Merging is repeated iteratively until both the two ends of the current polyline can not be extended any more.

As an illustration, Figure 5 presents the rectilinear polylines extracted from the graph of Figure 4. By excluding those lines belonging to the extracted connections, a symbol is isolated within the rectangular window.

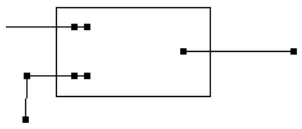


Figure 5: Extracted rectilinear polylines

Symbol matching: Once a loop-symbol is isolated, a six-tuple moment vector is computed from the constituent line segments. The symbol would be one of the models in the symbol class (associated with the matched characteristic loop) if the Euclidean distance between the corresponding moment vectors is less than a preset threshold.

4 Experimental results

The system has been implemented in C programming language on a SPARC–2 workstation. The input logic diagrams are digitized at a resolution of 150 pixels per inch.

Figure 6 shows a graph of line segments generated by the preprocessing module for a 800×1248 image of a logic diagram. All the matched loops along with their rectangular windows are also presented. In Figure 7, the recognized symbols are replaced with the corresponding models and the connecting polylines are aligned to beautify their appearances. The misrecognized (or rejected) objects remain in their line segments.

Processing time for the test drawing is 6.1 seconds: 4.9 seconds for preprocessing and 1.2 seconds for symbol recognition. We have observed that an A4-size (210×297 mm) logic diagram with an average complexity can be processed within 15 seconds.

Comparing the results with respect to the input logic diagram, 1 of 31 symbols and 1 of 53 connections are rejected. Another experiment with a set of logic diagrams shows that more than 99 % of symbols and more than 96 % of connections are correctly recognized.

5 Conclusion

We have proposed an automatic system which converts paper-based logic diagrams into a computer representation in terms of symbols and connections. The proposed system achieves a degree of recognition accuracy, efficiency, and flexibility. It can recognize about 99 % of loop-symbols and more than 96 % of rectilinear connections from a logic diagram. The preprocessing part of the system runs very fast with two bits per pixel for storage. In addition, prior knowledge about the input diagrams can be easily modified without rewriting the recognition program.

References

[1] H. Bunke, "Experience with Several Methods for the Analysis of Schematic Diagrams," *Proc. 6th ICPR*, pp. 710-712, 1982.

[2] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, 1973.

[3] Y. Fukada, "A Primary Algorithm for the Understanding of Logic Circuit Diagrams," *Pattern Recognition*, Vol. 17, pp. 125-134, 1984.

[4] M.K. Hu, "Visual Pattern Recognition by Moment Invariants," *IRE Trans. Information Theory*, Vol. 8, pp. 179-187, 1962.

[5] P.C.K. Kwok, "A Thinning Algorithm by Contour Generation," *Communications of the ACM*, Vol. 31, pp. 1314-1324, 1988.

[6] A. Okazaki, et al., "An Automatic Circuit Diagram Reader with Loop-structure-based Symbol Recognition," *IEEE Trans. PAMI*, Vol. 10, pp. 331-341, 1988.

[7] C.T. Zhan and R.Z. Roskies, "Fourier Descriptors for Plane Closed Curves," *IEEE Trans. Computers*, Vol. 21, pp. 269-281, 1972.

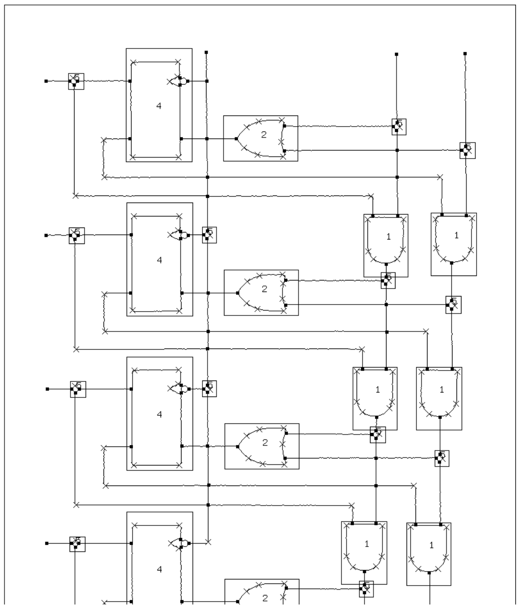


Figure 6: Preprocessed logic diagram

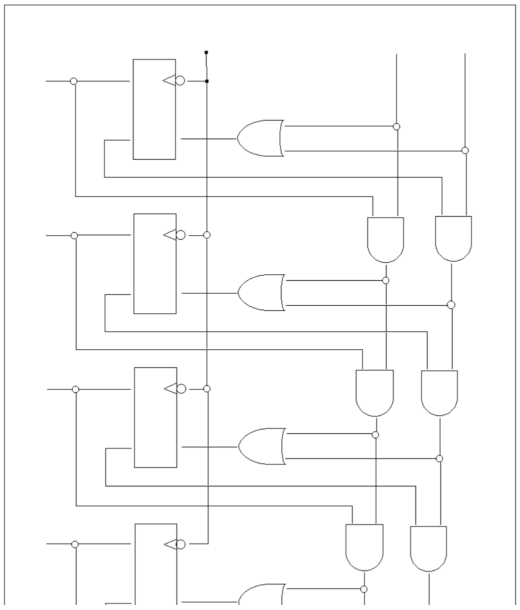


Figure 7: Recognition results