

Local Context-based Recognition of Sketched diagrams

Gennaro Costagliola, Mattia De Rosa, Vittorio Fuccella
Dipartimento di Informatica, University of Salerno
Via Giovanni Paolo II, 84084 Fisciano (SA), Italy
{gencos, matderosa, vfuccella}@unisa.it

Abstract

We present a new methodology aimed at the design and implementation of a framework for sketch recognition enabling the recognition and interpretation of diagrams. The diagrams may contain different types of sketched graphic elements such as symbols, connectors, text. Once symbols are distinguished from connectors and identified, the recognition proceeds by identifying the local context of each symbol. This is seen as the symbol interface exposed to the rest of the diagram and includes predefined attachment areas on each symbol. We argue that, in many cases, simple constraints on the local context of each symbol are enough to describe diagram languages defined on those symbols. Further refinement and interpretation of the set of acceptable diagrams is then provided through a visual grammar. We also describe the architecture of the framework and provide sample applications for the domains of flowcharts and binary trees.

Keywords: *sketch recognition, multi-domain, methodology, framework, visual languages*

1 Introduction

The use of diagrams is common in various disciplines. Typical examples include maps, line graphs, bar charts, engineering blueprints, architects' sketches, hand drawn schematics, etc. In general, diagrams can be created either by using pen and paper, or by using specific computer programs. These programs provide functions to facilitate the creation of the diagram, such as copy-and-paste, but the classic WIMP interfaces they use are unnatural when compared to pen and paper. Indeed, it is not rare that a designer prefers to use pen and paper at the beginning of the design [32], and then transfer the diagram to the computer later [2].

To avoid this double step, a solution is to allow users to sketch directly on the computer. This requires both specific hardware and sketch recognition based software. As

regards hardware, many pen/touch based devices such as tablets, smartphones, interactive boards and tables, etc. are available today, also at reasonable costs. Sketch recognition is needed when the sketch must be processed and not considered as a simple image and it is crucial to the success of this new modality of interaction. It is a difficult problem due to the inherent imprecision and ambiguity of a freehand drawing and to the many domains of application.

A central element of sketching is the stroke. On a touch screen, a stroke starts with the pressure of the pen (or finger) on the screen and ends when the pen is raised. Technically, a stroke is a finite list of triples (x, y, t) (or samples) where (x, y) are the pair of coordinates in which the pen was at the time t . The strokes are often preprocessed in order to extract the basic primitives from them.

In the literature we can find the description of many different frameworks for the recognition of diagrams. There are both solutions developed for specific domains [17, 13, 27] and multi-domain solutions [2, 4, 26, 18, 28, 14]. In the multi-domain frameworks, the low-level recognition is usually performed independently from the context, through the identification of graphical primitives (lines, arcs, ellipses, etc.). In the most advanced products, the domain knowledge is then used at a higher level, to correct possible low-level interpretation errors.

In this paper we present a new methodology aimed at developing a framework for the recognition of sketched diagrams from different domains. The main innovation regards the introduction of a recognition phase based on the analysis of the local context of symbols. This results to be effective since many visual languages need to be simple in order to be used, and as a result their structure happens to be simple enough to be captured with local checks. We prove this statement by showing that even a complex enough flowchart dialect can be fully syntactically modelled through this approach. From the point of view of sketch recognition one of the innovations introduced with this framework is that it learns directly from sample sketches of the specific domain the information used for low-level recognition, taking advantage of the various innovative machine learning-based

techniques produced in recent years (see next section).

The framework is logically composed of four layers: Text/Graphic Separation, Symbol Recognition, Local Context Detection and Diagram Recognition/Parsing. The first three layers are mostly pattern recognition processes to extract intermediate information from the strokes and to perform symbol and local context recognition. They include different modules to perform stroke segmentation, symbol identification in the diagram and the recognition of the attachment areas needed to connect the symbols to each other. The last layer consists of two modules. The first one is the Local Context-based Diagram Recogniser and validates the scanned diagram against simple well formedness rules. If validation succeeds then the diagram is recognised. The second one uses visual parsing techniques and is executed on the well formed diagram only if more checks and/or a syntactic interpretation are needed for further translation or execution of the diagram.

The rest of the paper is organised as follows: the next section contains a brief survey on frameworks for sketch recognition; in section 3 we describe the framework, its architecture and the main recognition techniques; section 4 gives the data to provide in order to instantiate the framework for a particular domain; lastly, some final remarks and a brief discussion on future work conclude the paper.

2 Related Work

In the literature of sketch recognition we can find the description of many solutions, both multi-domain or oriented to the interpretation of diagrams from specific domains (e.g., UML diagrams [3], electrical circuits [4], chemical drawings [5], etc.). In this brief survey we will only focus on the former frameworks. We will also briefly outline other proposals which only face specific subproblems (e.g., fragmentation of strokes, identification of primitives, recognition of symbols), since some of these techniques are used in our framework.

2.1 Multi-Domain Sketch Recognition

Most approaches exploit the domain knowledge to improve recognition at a lower level. *SketchREAD* [2] is a multi-domain sketch recognition framework which uses a structural description of the domain symbols to perform the recognition. The domain knowledge is also used in the low-level phases, in order to allow the system to recover from low-level recognition errors. *SketchREAD* was evaluated in two different domains: family trees and circuit diagrams.

AgentSketch [4] is a multi-domain sketch recognition system in which an agent-based system is used for interpreting sketched symbols. The method exploits the knowledge

about the domain context for disambiguating the symbols recognized at a lower level.

The framework presented in [26] exploits a combination of low level and high-level techniques to be less sensitive to noise and drawing variations. It has been evaluated on two domains: molecular diagrams and electrical circuits.

InkKit [5] is a sketch tool framework which works very similarly to ours. It firstly classifies the strokes as either writing or drawing, then identifies basic shapes such as lines, rectangles, and circles, then groups these primitives in text and diagram components and, lastly, identifies the relationships between components.

LADDER is a language [18] which enables the definition of sketched elements at different levels (e.g., primitives, symbols, entire diagrams, etc.) by giving a structural description of them, including components, geometric constraints, etc. The framework can automatically generate a domain specific sketch recognition system from each description and has been tested on many domains including UML diagrams, mechanical engineering, flowcharts and others.

Other frameworks working at a lower level than those cited above, but having possible application on a broad range of domains are *Paleosketch* [28] and *CALI* [14]. The former is a recognition and beautification framework that can recognize different classes of primitive shapes and combinations of them. The latter exploits a naive Bayesian classifier to recognize multi-stroke geometric shapes.

2.2 Low-Level Techniques for Sketch Recognition and Symbol Recognition

Some frameworks are aimed to the solution of specific subproblems of sketch recognition. In recent years we have seen notable improvements in low-level stroke processing and symbol recognition techniques. The most effective of them are machine learning-based and are aimed to: stroke fragmentation [35, 36, 27, 20, 19, 1] and recognition of unistroke [24, 29] and multi-stroke [31, 25, 21, 23, 16, 26, 33] symbols.

Stroke fragmentation is a very mature subfield of research in sketch recognition. It has produced interesting results in recent times, especially through the use of machine learning techniques. Its objective is the recognition of the graphical primitives composing the strokes and can be used for a variety of objectives, including structural symbol recognition [21, 12]. Most approaches break strokes in corners [35, 36, 27, 20], while some other approaches [19, 1] also use the so called *tangent vertices* (smooth points separating a straight line from a curve or parting two curves). Machine learning-based approaches are based on the extraction of some features from the points of stroke, particularly speed and curvature.

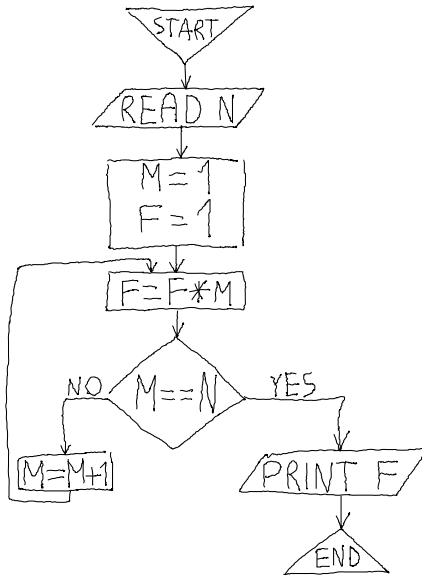


Figure 1: A simple flowchart for computing $N!$.

As for symbol recognition, the earliest methods [24, 29] were only able to recognize unistroke symbols. Several specialized methods have been recently proposed for multi-stroke hand-drawn symbol recognition. According to a widely accepted taxonomy [38] the methods are classified in two main categories: *structural* and *statistical*. In *structural* methods, the matching is performed by finding a correspondence between the structures, such as graphs [31, 25, 21] or trees [23], representing the input and the template symbols. In *Statistical* methods [16, 26, 33], instead, a given number of features are extracted from the pixels of the unknown symbol and compared to those of the models.

The recognition of unistroke symbols has had a recent progress and has been especially used in the recognition of gestures [34, 22, 9] for various applications, including interfaces for mobile devices [37, 15].

3 The Framework Design

The objective of the framework is to enable the recognition of diagrams from a wide range of domains. A common feature of these domains is the presence of three different types of graphics: symbols, connectors and text.

The framework has a layered architecture composed of the following four Layers, further divided into modules: *Text/Graphic Separation Layer*; *Symbol Recognition Layer*; *Local Context Detection Layer*; *Diagram Recognition/Parsing Layer*.

In the following we will refer to the flowchart in Figure 1 to exemplify the operations performed by the different layers.

3.1 The Text/Graphic Separation Layer

The scope of this layer is to separate freehand drawing (graphics) from handwriting (text) (see Figure 2b). This preliminary operation is necessary because text and graphics need to be managed separately. For the realization of this layer we relied on the technique presented in [3]. For sake of conciseness, in the rest of the paper we focus on the graphic aspects of the diagram.

3.2 The Symbol Recognition Layer

This layer recognizes the user drawn sketched symbols. It is further divided in the following modules:

- **Stroke Preprocessing Module.** This module identifies the graphical primitives present in the graphic domain of the diagram (see Figure 2c). This is done in two distinct phases: a *segmentation* phase, in which a stroke is divided in more primitives and a *clustering* phase in which different segments are put together to form a primitive. Segmentation is performed by detecting corners through *RankFrag* [6], a novel technique derived from previous machine learning-based methods [27, 20].
- **Symbol Identification Module.** This module clusters the primitives identified at the previous step in two different classes: *symbols*, *connectors* (see Figure 2d). For the realization of this module we relied on the technique, based on machine learning, described in [30].
- **Symbol Recognition Module.** Once the primitives composing a symbol have been grouped together, the symbol must be assigned to a class of known symbols (see Figure 2e). The recognition task is performed by this module by using the technique described in [8], which is a point cloud technique invariant with respect to scaling and supports symbol recognition independently from the number and order of strokes.

3.3 The Local Context Detection Layer

In the last years many methods to model a diagram as a member of a visual language have been devised by researchers. Basically, a diagram has been represented either as a set of relations on symbols (the *relation-based approach*) or as a set of attributed symbols with typed attributes representing the “position” of the symbol in the sentence (the *attribute-based approach*) [11]. Even though the two approaches appear to be very different, they both model a diagram (or visual sentence) as a set of symbols and relations among them. Differently from the relation-based

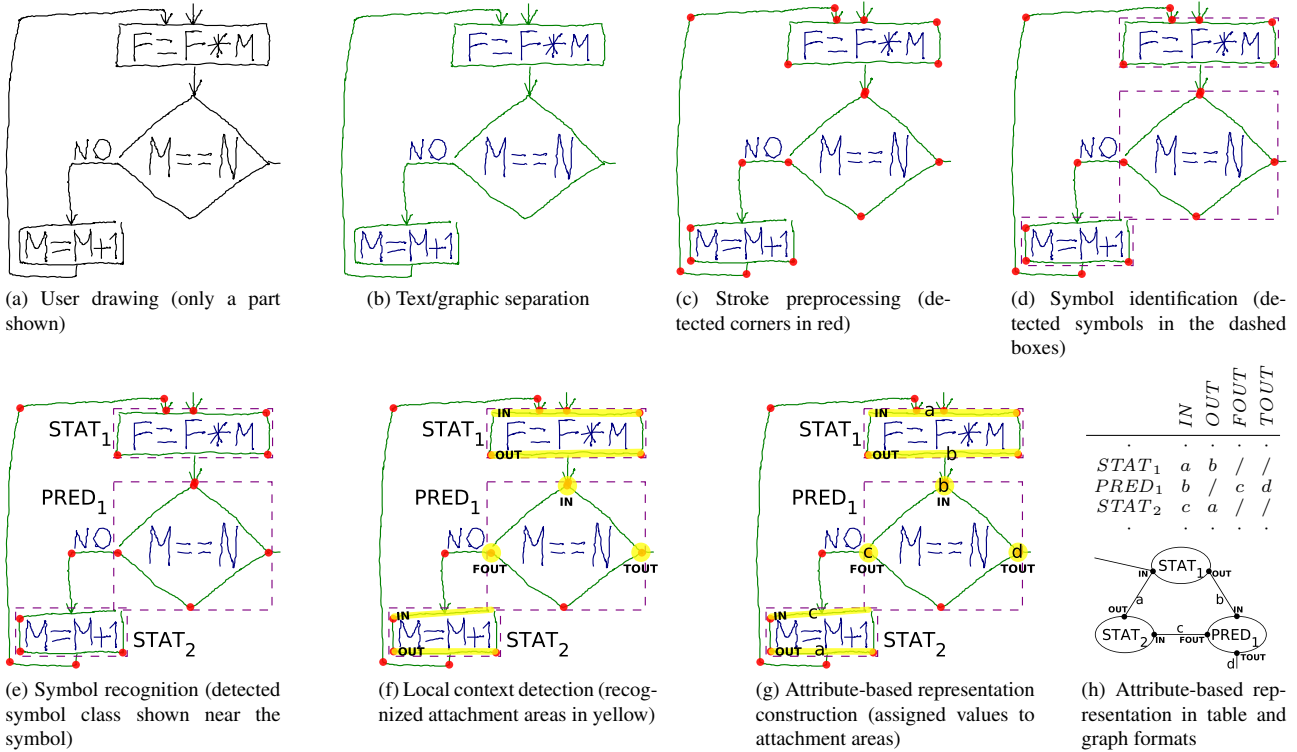


Figure 2: Recognition steps for a part of the flowchart in Figure 1.

approach where the relations are explicit, in the attribute-based approach the relations are implicit and must be derived by associating attribute values. The former approach is therefore at a higher level with respect to the latter. In this paper we adopt the attribute-based approach since it allows us to work at the lowest level possible. Moreover, we define the local context of a symbol as the set of its typed attributes. These are specified together with the visual characteristics of a symbol at definition time and are derived at recognition time from the way the symbol interacts with other symbols in a diagram. As an example, the rhomboid symbol in Figure 1 is defined in Table 1 with three attributes typed as *IN* (input), *FOUT* (output-if-false) and *TOUT* (output-if-true), visually corresponding to the three attaching points of the symbol. The first column of Table 1 shows the definition of flowchart symbol identifiers together with their typed attributes while the second column gives information about the symbol visual aspect and the location of its attributes. (The remaining columns of the Table will be described in the next section). The goal of the Local Context Detection Layer is then to identify the attributes and their types for each symbol (see Figure 2f). In our system, the attributes are identified by using an approach similar to the one proposed in [7]. The approach is independent from the method used to recognize symbols and assumes that the

Token	Graphics	Token occurrences	Constraints
BEGIN: <i>OUT</i>		1	$ OUT = 1$
END: <i>IN</i>		1	$ IN \geq 1$
STAT: <i>IN</i> , <i>OUT</i>		≥ 0	$ IN \geq 1$, $ OUT = 1$
IO: <i>IN</i> , <i>OUT</i>		≥ 0	$ IN \geq 1$, $ OUT = 1$
PRED: <i>IN</i> , <i>FOUT</i> , <i>TOUT</i>		≥ 0	$ IN \geq 1$, $ FOUT = 1$, $ TOUT = 1$

Table 1: Flowchart symbol specifications.

symbol has already been recognized.

3.4 The Diagram Recognition/Parsing Layer

This layer is composed of two modules: the Local Context-based Recogniser and the Diagram Parser. They

execute sequentially and in some uses of the framework, when the language is fully specified by the local constraints or in the case of fast prototyping, only the first module is needed.

3.4.1 Local Context-based Recogniser

The output of the recogniser is the attribute-based representation of the diagram if the diagram is well formed. This representation is constructed by giving values to the symbol attributes identified in the previous layer while checking for the well formedness of the diagram against simple constraints. In order to give values to attributes, the module generates value ids and assigns them such that two attributes have the same value only if the corresponding graphical counterparts are connected (see Figure 2g). This will then produce the attribute-based representation of the diagram (shown in Figure 2h both in tabular and graph formats). As for well formedness, constraint rules are given at definition time together with the symbol specifications and are intended to be easy to verify. In our example, the third and fourth columns of Table 1 indicate how many times a symbol may occur in a diagram and the number of values that an attribute may have, respectively. The symbol PRED may then appear zero or more times in a diagram and may have multiple input connections but only one exiting connection for each output attaching point. In our experience, simple constraints such as the ones above are enough to completely describe a visual language. In the case of the flowcharts as the one depicted in Figure 1, it is easy to verify that Table 1 together with the three rules “each connection must be 1-to-1”, “each IN attaching point must only be connected to an (F/T)OUT attaching point” and “the attribute-based (graph) representation must be connected” completely specify a set of flowcharts that is Turing-complete.

The Local Context-based Recogniser takes as input an XML specification file where all the rules about tokens and connections are coded. Figure 3 shows the XML file specification for the case described above coding the information in Table 1 and the three additional rules. In the XML specification, each table row is defined through a token element. The name of the token, the file containing its graphical representation, and the number of its occurrences in a language instance are defined through the name, ref and occurrences attributes, respectively. Each ap element defines one of the token attaching points by specifying its type (type attribute), name (name attribute), a reference to its position in the graphical representation (ref attribute) and the number of allowed connection (connectNum attribute). The constraint “the attribute-based (graph) representation must be connected” is specified by <constraint>connected</constraint>.

```
<language name="flowchart">
  <token name="begin" ref="triangleDown.svg"
    occurrences=="=1">
    <ap type="exit" name="out" ref="lowPoint"
      connectNum=="=1" />
  </token>
  <token name="end" ref="triangleUp.svg"
    occurrences=="=1">
    <ap type="enter" name="in" ref="hiPoint"
      connectNum=="=1" />
  </token>
  <token name="stat" ref="rectangle.svg"
    occurrences=">=0">
    <ap type="enter" name="in" ref="hiLine"
      connectNum=">=1" />
    <ap type="exit" name="out" ref="lowLine"
      connectNum=="=1" />
  </token>
  <token name="io" ref="parallelogram.svg"
    occurrences=">=0">
    <ap type="enter" name="in" ref="hiLine"
      connectNum=">=1" />
    <ap type="exit" name="out" ref="lowLine"
      connectNum=="=1" />
  </token>
  <token name="pred" ref="rhombus.svg"
    occurrences=">=0">
    <ap type="enter" name="in" ref="hiPoint"
      connectNum=">=1" />
    <ap type="exit" name="fout" ref="leftPoint"
      connectNum=="=1" />
    <ap type="exit" name="tout" ref="rightPoint"
      connectNum=="=1" />
  </token>
  <connector ref="arrow">
    <cap type="enter" ref="head" connectNum=="=1"
      />
    <cap type="exit" ref="tail" connectNum=="=1"
      />
  </connector>
  <constraint>connected</constraint>
</language>
```

Figure 3: Flowchart specification.

The connector element describes how tokens are connected. It defines its type (from a predefined list of implemented types) and specifies that the head of the arrow must be connected to an enter attaching point, while the tail must be connected to an exit attaching point. The predefined type arrow together with the two connectNum=="=1" conditions guarantee that the property “each connection must be 1-to-1” is satisfied, while the use of the type definitions enter and exit in the token elements guarantees that the property “each IN attaching point must only be connected to an F/T)OUT attaching point” is satisfied.

As a second example of application, let us now consider the language of the binary trees. In this case, the symbol specification shown in Table 2 and the three constraints “each connection must be 1-to-1”, “each IN attaching point



Token	Graphics	Token occurrences	Constraints
ROOT: <i>IN</i> , <i>OUT</i>		1	$ IN = 0,$ $ OUT \leq 2$
NODE: <i>IN</i> , <i>OUT</i>		≥ 0	$ IN = 1,$ $ OUT \leq 2$

Table 2: Binary tree symbol specifications.

```

<language name="binaryTree">
  <token name="root" ref="circle.svg" occurrences
    ==1">
    <ap type="enter" name="in" ref="hiSC"
      connectNum=="=0" />
    <ap type="exit" name="out" ref="lowSC"
      connectNum="<=2" />
  </token>
  <token name="node" ref="circle.svg" occurrences
    ==>0">
    <ap type="enter" name="in" ref="hiSC"
      connectNum=="=1" />
    <ap type="exit" name="out" ref="lowSC"
      connectNum="<=2" />
  </token>
  <connector ref="polyline">
    <cap type="exit" ref="p0" connectNum=="=1" />
    <cap type="enter" ref="p1" connectNum=="=1" /
  >
  </connector>
  <constraint>connected</constraint>
</language>

```

Figure 4: Binary tree specification.

*must only be connected to an OUT attaching point” and “the attribute-based (graph) representation must be connected”, as coded in the XML specification shown in Figure 4, completely describe the language. Here, ROOT and NODE have the same graphical representation and are only distinguished for the number of occurrences and the constraints on the *IN* attaching point.*

3.4.2 Diagram Parser

This parser is built only if a syntactic interpretation of the diagram is needed for further processing, such as translation or execution, and/or if the language cannot be completely specified by a set of simple constraints. This is similar to the division of roles between the lexical and syntactic phases for a traditional compiler. The diagram parser takes as input the attribute-based representation produced in the previous module and a visual grammar for the syntax specification. Many visual grammar formalisms and corresponding pars-

ing algorithms may be found in literature and, even though we adopt a parsing technique based on the principles described in [10], the framework is not linked to any specific type of visual parser technology. Moreover, it is important to note that, since the input to the parser is already well formed, the complexity of this module is simplified with respect to other approaches.

In order to show a case when the local context recognition needs to be complemented by a syntax analysis phase let us consider a structured version of the flowchart language described in the previous section (see Figure 5). To structure the language we introduce two more tokens with names B-BEGIN and B-END whose roles are the same as the block delimiters “{” and “}” in the C language, respectively. Each of the two tokens is specified similarly to the token STAT with number of occurrences ≥ 0 and two attaching points *IN* and *OUT* with types *enter* and *exit*, respectively, and $|IN| \geq 1$ and $|OUT| = 1$. As in any structured language, the block delimiters B-BEGIN and B-END are to be used in pairs and then other rules should be added. As already known, these are not constraints that can be solved by locally looking at the properties of a single token. As a consequence, the technique described in the previous section cannot be used to capture the whole structured flowchart language. We now provide a visual grammar describing a limited structured flowchart language including the flowchart in Figure 5. The grammar is composed by a set of terminals given by the tokens as described in Figure 3 in the format: TOKEN(*attaching_point1*, *attaching_point2*, ...), a set of non terminals in the same token format: Nterm(*attaching_point1*, *attaching_point2*, ...), an initial terminal FlowCh, and the set of productions shown in Figure 6. In each production, the single letters x, y, u, ... represent, when in the right part of the production, connections between token and/or non terminal attaching points. When in the left part of a production, they indicate which attaching points of the subsentence are externally exposed. The notation $x \uplus z$ indicates that the two attaching points marked by x and z will be connected to the same target attaching point. As an example, the subsentence in Figure 2g matches and instantiates production 7 as follows: Block(a, d) \rightarrow Block(a,b) PRED(b, c, d) Block(c, a) where Block(a,b) comes from matching STAT₁ with production 5 instantiated as Block(a, b) \rightarrow STAT(a, b) and Block(c, a) comes from matching STAT₂ with production 5 instantiated as Block(c, a) \rightarrow STAT(c, a). As already pointed out, in the literature there are many approaches that use visual grammar formalisms, at least as powerful as the one above, to generate visual parsers directly from a grammar.

It can be noted that, without a local context analysis, syntax errors such as adding an extra connection between any token in Figure 5, cannot be easily detected by only using a grammar approach.

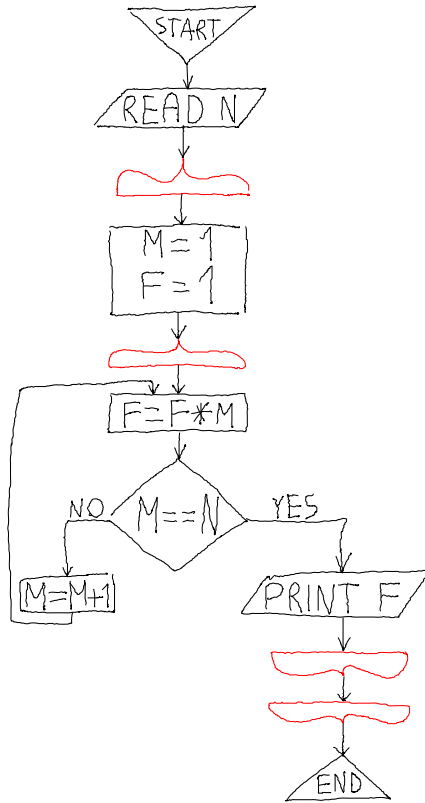


Figure 5: A simple structured flowchart containing the B.BLOCK and B.END tokens.

4 Instantiating the Framework

In order to instantiate the framework for a specific domain, it is necessary to provide input data. In particular, being mostly based on machine learning, the Symbol Recognition Layer modules need a training phase, while the higher level layers need formal definitions. The following data must be provided:

1. Sample diagrams from the domain, with the following annotations:
 - Types of strokes (text, graphics);
 - Strokes clustering/segmentation;
 - Connector/symbols separation;
 - Class names of the symbols;
 - Attachment areas on the symbols;
2. The XML specification of the language and the referenced files with the graphical definitions of the symbols and attaching points;
3. The specification for the syntax interpretation of a diagram (optional).

1. FlowCh \rightarrow BEGIN(x) Block(x, y) END(y)
2. Block(x, y) \rightarrow Block(x, z) Block(z, y)
3. Block(x, y) \rightarrow B_BEGIN(x, z) Block(z, u) B_END(u, y)
4. Block(x, y) \rightarrow IO(x, y)
5. Block(x, y) \rightarrow STAT(x, y)
6. Block(x, y \cup z) \rightarrow PRED(x, u, v) Block(u, y) Block(v, z)
7. Block(x \cup z, y) \rightarrow Block(x, u) PRED(u, v, y) Block(v, z)
8. Block(x \cup z, y) \rightarrow PRED(x, u, y) Block(u, z)

Figure 6: Visual grammar describing a structured flowchart language.

4.1 Implementation

The framework is being developed in Java.

In addition to the development of the modules for the recognition, we are working at the development of an environment that facilitates the production of input data needed to instantiate the framework for a particular domain. Specifically, we're providing a GUI for making quick annotations on the sample diagrams and to define the constraints and the syntax specification.

5 Conclusions

We described a local context-based recognition methodology whose final objective is the development of a framework for multi-domain sketch recognition and interpretation. The diagrams may contain different types of graphic elements (symbols, connectors, text). Future work will include the instantiation of the framework in different domains. At the end of the development phase, we will perform tests to evaluate the effectiveness and efficiency of the individual modules and of the overall framework. We will make comparative evaluations with state-of-art techniques.

References

- [1] F. Albert, D. Fernandez-Pacheco, and N. Aleixos. New method to find corner and tangent vertices in sketches using parametric cubic curves approximation. *Pattern Recognition*, 46(5):1433 – 1448, 2013.
- [2] C. Alvarado and R. Davis. Sketchread: a multi-domain sketch recognition engine. In *Proc. of UIST '04*, pages 23–32, 2004.
- [3] D. Avola, A. Buono, P. Nostro, and R. Wang. A novel online textual/graphical domain separation approach for sketch-based interfaces. In E. Damiani, J. Jeong, R. Howlett, and L. Jain, editors, *New Directions in Intelligent Interactive Multimedia Systems and Services - 2*, volume 226, pages 167–176. Springer, 2009.

- [4] G. Casella, V. Deufemia, V. Mascardi, G. o Costagliola, and M. Martelli. An agent-based framework for sketched symbol interpretation. *JVLC*, 19(2):225–257, 2008.
- [5] R. Chung, P. Mirica, and B. Plimmer. Inkkit: A generic design tool for the tablet pc. In *Proc. of CHINZ'05*, pages 29–30, 2005.
- [6] G. Costagliola, M. De Rosa, V. Fortino, and V. Fucella. Rankfrag: A novel machine learning-based technique for hand-drawn sketch segmentation. Submitted for publication, Apr. 2014.
- [7] G. Costagliola, M. De Rosa, and V. Fucella. Identifying attachment areas on sketched symbols. In *Proc. of VL/HCC '11*, pages 83–86, 2011.
- [8] G. Costagliola, M. De Rosa, and V. Fucella. Improving shape context matching for the recognition of sketched symbols. In *Proc. of DMS*, pages 289–294, 2011.
- [9] G. Costagliola, M. De Rosa, and V. Fucella. Investigating human performance in hand-drawn symbol autocompletion. In *Proc. of SMC '13*, pages 279–284, 2013.
- [10] G. Costagliola, V. Deufemia, and M. Risi. Sketch grammars: a formalism for describing and recognizing diagrammatic sketch languages. In *Proc. of ICDAR'05*, pages 1226–1230, 2005.
- [11] G. Costagliola and G. Polese. Extended positional grammars. In *Proc. of VL '00*, pages 103–110, 2000.
- [12] G. Costagliola, M. D. Rosa, and V. Fucella. Recognition and autocompletion of partially drawn symbols by using polar histograms as spatial relation descriptors. *Computers & Graphics*, 39(0):101–116, 2014.
- [13] G. Feng, C. Viard-Gaudin, and Z. Sun. On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming. *Pattern Recognition*, 42(12):3215–3223, 2009.
- [14] M. Fonseca and J. Jorge. Using fuzzy logic to recognize geometric shapes interactively. In *Proc. of FUZZ'IEEE*, volume 1, pages 291–296 vol.1, 2000.
- [15] V. Fucella, M. De Rosa, and G. Costagliola. Novice and expert performance of keystretch: a gesture-based text entry method for touch-screens (in press). *IEEE Transactions on Human-Machine Systems*, 2014.
- [16] L. Gennari, L. B. Kara, T. F. Stahovich, and K. Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics*, 29(4):547–562, 2005.
- [17] T. Hammond and R. Davis. Tahuti: a geometrical sketch recognition system for uml class diagrams. In *ACM SIGGRAPH 2006 courses*, 2006.
- [18] T. Hammond and R. Davis. Ladder, a sketching language for user interface developers. In *ACM SIGGRAPH 2007 courses*, 2007.
- [19] J. Herold and T. F. Stahovich. Speedseg: A technique for segmenting pen strokes using pen speed. *Computers & Graphics*, 35(2):250–264, 2011.
- [20] J. Herold and T. F. Stahovich. A machine learning approach to automatic stroke segmentation. *Computers & Graphics*, 38(0):357–364, 2014.
- [21] W. Lee, L. Burak Kara, and T. F. Stahovich. An efficient graph-based recognizer for hand-drawn symbols. *Computers & Graphics*, 31:554–567, August 2007.
- [22] Y. Li. Protractor: A fast and accurate gesture recognizer. In *Proc. of CHI '10*, pages 2169–2172, 2010.
- [23] Y. Lin, L. Wenyn, and C. Jiang. A structural approach to recognizing incomplete graphic objects. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 1, pages 371–375 Vol.1, aug. 2004.
- [24] J. S. Lipscomb. A trainable gesture recognizer. *Pattern Recognition*, 24:895–907, September 1991.
- [25] J. Lladós, E. Martí, and J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Trans. PAMI*, 23(10):1137–1143, Oct. 2001.
- [26] T. Y. Ouyang and R. Davis. A visual approach to sketched symbol recognition. In *Proc. of IJCAI'09*, pages 1463–1468, 2009.
- [27] T. Y. Ouyang and R. Davis. Chemink: a natural real-time recognition system for chemical drawings. In *Proc. of IUI '11*, pages 267–276, 2011.
- [28] B. Paulson and T. Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *Proc. of IUI '08*, pages 1–10, 2008.
- [29] D. Rubine. Specifying gestures by example. *SIGGRAPH Comput. Graph.*, 25:329–337, July 1991.
- [30] T. F. Stahovich, E. J. Peterson, and H. Lin. An efficient, classification-based approach for grouping pen strokes into objects. *Computers & Graphics*, (0):–, 2014.
- [31] W.-H. Tsai and K.-S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Trans. Systems Man Cyber.*, 9(12):757–768, dec. 1979.
- [32] D. G. Ullman, S. Wood, and D. Craig. The importance of drawing in the mechanical design process. *Computers & Graphics*, 14(2):263–274, 1990.
- [33] D. Willems, R. Niels, M. van Gerven, and L. Vuurpijl. Iconic and multi-stroke gesture recognition. *Pattern Recognition*, 42(12):3303–3312, 2009. New Frontiers in Handwriting Recognition.
- [34] J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a \$1 recognizer for user interface prototypes. In *Proc. of UIST '07*, pages 159–168, 2007.
- [35] A. Wolin, B. Eoff, and T. Hammond. Shortstraw: A simple and effective corner finder for polylines. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*. Eurographics Association, 2008.
- [36] Y. Xiong and J. J. J. LaViola. A shortstraw-based algorithm for corner finding in sketch-based interfaces. *Computers & Graphics*, 34(5):513–527, 2010.
- [37] S. Zhai and P. O. Kristensson. The word-gesture keyboard: Reimagining keyboard interaction. *Commun. ACM*, 55(9):91–101, Sept. 2012.
- [38] W. Zhang, L. Wenyn, and K. Zhang. Symbol recognition with kernel density matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(12):2020–2024, dec. 2006.