Grammar-Driven Interpretation of Engineering Drawings

S.H. Joseph T.P. Pridmore M.E. Dunn

Dept. of Mechanical and Process Engineering, University of Sheffield

We describe a methodology and associated system for the interpretation of images of engineering drawings based on the exploitation of the Unix tool Yacc. The system is extensible and clearly relates a hypothesis/prediction/verification strategy to a hierarchical model description and the deployment of low level techniques. Examples are given of the interpretation of real mechanical engineering drawings.

1 Introduction

The automatic analysis of line drawings has been approached from various viewpoints. Sugihara (1986) and others consider line drawings to be those descriptions that might arise after perfect edge extraction from an image of a polyhedral scene, and attempt to reconstruct that scene. Haralick and Queeney (1982) and others consider the problem of reconstructing a 3D object from an idealised engineering drawing of its orthogonal views. Lin and Pun (1978) and others examine input sketched on a digitising tablet and attempt to interpret it, aiming to provide a superior human interface to CAD systems. Mackworth et al (Mackworth 1977, Mulder, Mackworth and Havens 1987) have taken their MAPSEE system for applying constraints to sketch map interpretation through several stages of development. Also in sketch analysis, MIRABELLE (Masini and Mohr 1983, Mohr 1986) employs precompilation and bi-directional parsing of a picture description language. Although the latter two projects are of particular interest, none of the above deal with images directly, as we do here.

Turning to the processing of real images of real documents, there is an extensive literature on the extraction of line structured data (eg Smith 1987, Black et al 1981) and on text and symbol recognition in diagrams (Lin et al 1985, Tudhope and Oldfield 1983, Bley 1984, Bunke 1982).

This work has been stimulated in part by the widespread introduction of computer based draughting systems: industries with large stocks of conventional engineering drawings demand conversion from the old format to the new. Commercially, this demand has been met partly by manual conversion (with CAD system support) and partly by automatic scanning systems (with interactive post-processing) (Hofer-Alfeis 1986). The conversion process involves the extraction of line structured data from an image and its interpretation and composition into entities found in CAD systems (polylines, lines of various styles, symbols, text, curves, hatching, etc). The automation of this process is the goal of the project of which the present work forms a part.

Our earlier work on line drawing conversion (Joseph 1988) has shown that direct, sequential extraction of simple primitives (straight lines) from the original image provides both line enhancement and opportunities for the interpretation of elaborate linework. Furthermore, the extraction of compound entities probably calls for their being directly sought as well. The complexity of this task given useful A0 size engineering drawings is severe. We require some means of describing the entities sought, and would prefer to use this description to drive the search. The drawings themselves are (nominally) produced to a strict convention, though this may vary from one engineering discipline to another. Our description should relate transparently to such a convention, and should show how the entities are composed of primitive elements. A grammatical description with associated parser is a natural approach.

2 Yacc: A Parser Generator

Much work on syntactic pattern recognition starts from the grammar as a descriptive tool, validates its generative power, then deals with parsing, or transformation into parsable form. The size and complexity of the drawing conversion problem means that our approach has to be rather the opposite: we seek a tractable parsing procedure, and examine the possibility of applying it to the task in hand. The aim is to find out how much of the problem can be clearly and efficiently solved by these means, not to prove that the chosen grammar is powerful enough to generate a particular class of line drawings. This may seem an excessively pragmatic approach, but it has more justification than mere expediency. It foregoes the freedom to construct a grammar as intuition and experience will, but gains the security of a transparent parsing process. It is otherwise all too easy to sacrifice intelligibility of the parsing process to the demands of a rigourous grammar.

The unix yacc (yet another compiler compiler) utility

AVC 1988 doi:10.5244/C.2.36

(Johnson 1975) is a parser generator usually employed to create command interpreters and language compilers. It accepts an LR(1) grammar together with action code in the C language (to build parse trees for example) and generates a table driven, finite state machine. This calls a user-supplied function to obtain the tokens (terminals) to be parsed and invokes the appropriate action code whenever a grammar rule is reduced. LR(1) grammars generate token strings that can be deterministically parsed in a left to right scan with a single token lookahead. There is no backtracking.

Each yacc token is associated with a value (which may be a pointer to a complex data structure). These are made available to the action code; should that code generate new information it can pass it on through the value associated with the left hand side of the rule. The yacc machine is stack-based, each stack element containing the current state number and a value. Yacc can provide a listing of the states of the machine together with the rules, marked as to their extent of completion, associated with each state. A further, and perhaps more valuable, debugging aid is yacc's ability to detect and describe inconsistencies in the rule set during generation of the parser.

Yacc provides a rapid, compact and portable method of syntactic analysis which may be applied straightforwardly in syntactic recognition methods based on a suitable string grammar (Henderson and Samal 1986). However, its application to the drawing conversion problem cannot be immediate: a token stream must somehow be extracted from the input image.

3 Yacc in Drawing Analysis

Our methodology for token extraction owes much to Joseph's (1988) work on sequential line following. As part of that work, a line tracking procedure was developed for extracting a substantial length of straight linework from a grey level image of a drawing. Processing was enhanced by tracking subsequent child lines from a search at each end of a parent line, permitting dynamic thresholding based on the parent's properties and the tracking of lines through areas of complex interaction. Tracking thus spreads in a treelike fashion from an initial start point. Examination of the system in operation suggests that certain higher level entities might be tracked as temporally ordered, linear sequences of line primitives connected by appropriate geometrical relations. These sequences can be expressed as rules in a suitable grammar. Appendix 1, for example, shows a simple grammar for drawings consisting only of physical outlines (solid segments joined by corners) and isolated broken lines.

It is important to stress here that the proposed use of the yacc grammar is more to control the basic line follower than to define what it is to be an engineering drawing. LR(1) grammars are inherently one-dimensional and so cannot easily be used to describe 2D drawings in any declarative sense. It is possible, however, to specify drawing analysis strategies using a string grammar formalism; rules represent ordered sequences of observations. Rule 3 in appendix 1 should therefore be read as a statement that to obtain a broken line segment one must first locate a line primitive, then note a suitable break relation specifying the (relative) location of a second line. This is not as large a departure from traditional use of grammatical techniques as it might at first appear to be. A string grammar may specify how to generate a set of engineering drawings procedurally, just as an equivalent 2D grammar would define them declaratively.

Once a higher level entity is identified it is clearly desirable that the tracking procedure be modified to explicitly seek the next primitive associated with that entity: this might permit the extraction of higher constructs from complex and noisy linework, a very desirable result. The tracker is tuned in this way by allowing the yacc machine to pass the line following module a pointer to the top of its value stack, making available the value of the last token received or the left hand side of the last rule reduced. An appropriate search strategy may then be determined. As stack values reference different constructs at different points in the parse an object-oriented approach (Cox 1986) has been taken. On receipt of the value parameter the line follower calls functions belonging to the current object to pursue the tracking and calculate the token to be returned to the parser. Line tracking and token classification functions effectively 'belong' to specific entities, and form a frame of reference for developing those objects.

Token extraction therefore takes place in the object at the top of the stack. This method of handling context is both complete, in that the stack can reference all the information so far found, and practical, in that the top of the stack represents the most immediate context. At present only two stack elements are ever considered. If the top of the stack contains a drawing construct this is solely responsible for calling (in object-oriented fashion) the appropriate token extraction procedures. Some objects, however, describe geometrical relations between entities; jumps or changes of direction in the tracking such as occurs between dashes on a broken line, between lines of crosshatching, or around a corner in a physical outline. When this type of object (eg. BREAK, CORNER in appendix 1) is called on to continue the tracking it passes the task on to the last drawing construct down the stack, together with information about the transformation. The object oriented approach facilitates this.

Note that the two-dimensional movements of the line tracker are implicit in the token extraction and so are controlled by individual objects rather than by the grammar. The parser does, however, exert some directional control when several alternative search paths are available, in which case we seek a token that satisfies the grammar. This is readily done by applying the yacc machine's error checking code to each available token before returning one to the parser: erroneous tokens are rejected if one acceptable to the grammar can be generated from

an alternative tracking path. Should more than one valid token be available an object-oriented selection function is invoked.

Consider the simple rule set of appendix 1 applied to the data of figure 1a. At the beginning of the parse an empty object corresponding to the start symbol is placed on the stack. That object controls the line tracker: any piece of linework found will cause it to return the token LINE. The machine then reduces rule 7 and the action code instantiates a line whose name and address is placed on the stack. The next tracking and token generation phase is controlled by that line, which returns a BREAK token when its beam-like set of circular searches (fig 1b) discover the gap between the end of the first line and the start of the second. A break object is now on top of the stack. As it describes a relation rather than a drawing construct it is passed down the stack as a parameter to the previous line object, which uses knowledge of itself and the break relation to generate a search for the next piece of linework. A circular search of appropriate radius is performed at the expected position of the next line end (not shown) and a second line token returned. Rule 7 is invoked again and the LINE terminal replaced by an instantiated line. Now that a "line BREAK line" sequence has been detected rule 3 is reduced and a broken line object instantiated. This replaces its component parts on the stack and will control subsequent searches.

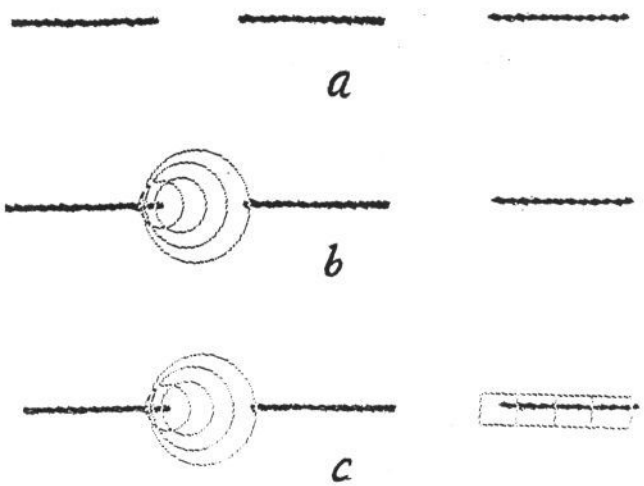


Figure 1. The interpretation of a single dashed line. a) The thresholded image. The tracker's user-defined start point is marked by a cross. b) A beam-like set of circular searches detect a break in the line. c) After instantiation of a broken line subsequent segments are predicted in top-down fashion and tested by linear search patterns.

Although LR parsers are essentially bottom-up devices, the addition of object-oriented techniques allows top-down control of image processing and token extraction functions by higher level constructs. To continue our example, the bottom-up instantiation of the broken.line object described above provides (hypothesised) higher level context which may be used to guide the search. The broken line now estimates the position, length and width of the next line in the sequence and generates a set of linear search patterns to test its prediction in top-down fashion (figure 1c). Should an appropriate line be found, related to the broken line by a suitable break, rule 4 is reduced and the construct extended. Hence the hierarchical rule structure provides a framework for the integration of bottom-up and top-down processing.

As figure 1 shows, objects invoke search patterns according to context; all those seen thus far seek straight line segments. There are, however, drawing constructs for which the line element is less than ideal; a detailed description of text, for example. Although the text object is triggered by the detection of short lines at acute angles, words are tracked letter by letter. Each letter is detected by edge tracking a black area of suitable size and location. In this way the parser becomes an excellent means for deploying appropriate extraction techniques as new data (or context) becomes available.

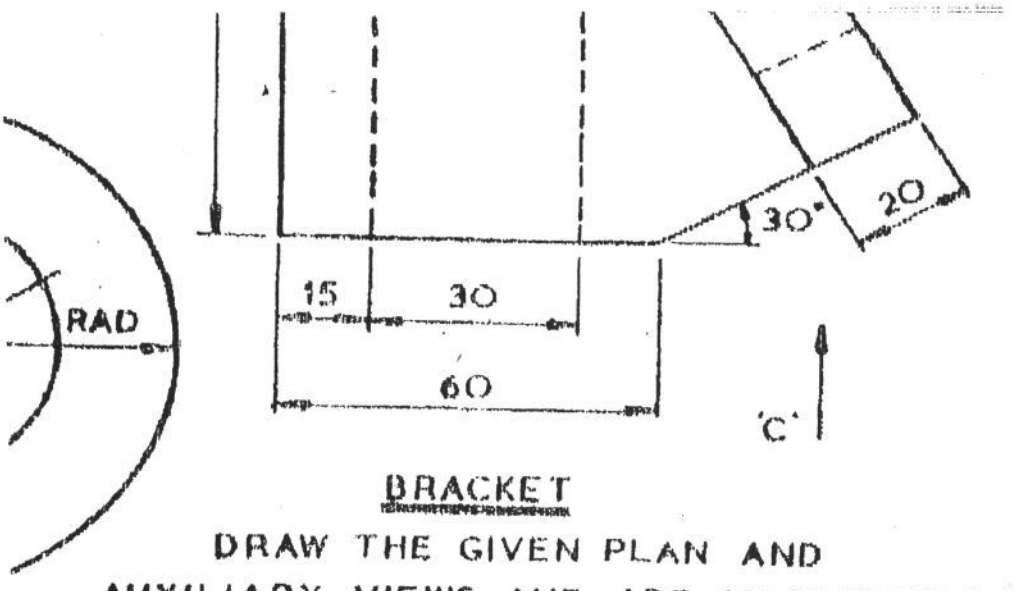
4 Results

The grammar now comprises 52 rules resulting in a 78 state machine covering continuous and broken lines, curves, characters, words, solid and broken crosshatching. A thresholded drawing image is shown in figure 2a. Crosshatched regions obtained from operator supplied start positions are shown in figure 2b. The results of running the system from repeated machine generated search positions are shown in figure 3. No attempt is made to merge constructs so some may be found more than once.

5 Discussion

The system presented here may be thought of as an "entity extractor", applying user-defined search strategies to find common drawing constructs. The structure of the grammar allows these to be defined as hierarchical combinations of primitive elements. Drawing entities are therefore not treated in isolation. Instead the parser has knowledge, compiled by yacc from the rule hierarchy, of all relevant interactions between the drawing entities represented within the grammar. This allows consideration of all available possibilities at each step, leading to efficient search and enabling the yacc machine to detect and report illegal token sequences.

Although worthwhile in itself, the present approach should be placed in perspective as an example of model-based image analysis. There are two identifiable research groupings in the literature, which we will refer to as AI (those who use terms such as rule, inference engine, uncertainty, consistency maintenance) and syntactic pattern recognition (who refer to grammar rules, parsers, errors and error correction).

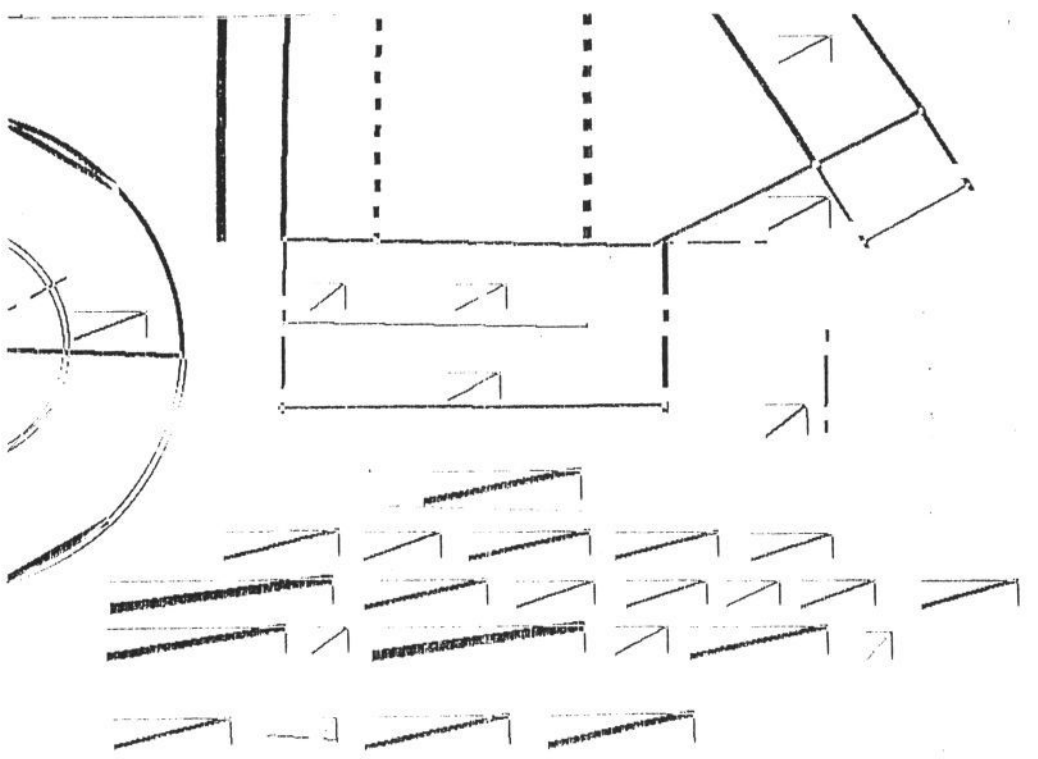


BRACKET

DRAW THE GIVEN PLAN AND
AUXILIARY VIEWS AND ADD AN ELEVATION
LOOKING IN DIRECTION OF ARROW 'C'

SHOW ALL HIDDEN DETAIL.

a



b

Figure 2. a) A thresholded image of a mechanical engineering drawing. b) The output of the grammar-driven system, showing text as diagonal thick lines inside half-boxes, chain lines, curves, broken and continuous lines.

In a survey of AI related literature, Binford (1982) remarks on the need for more powerful low level image processing operations, the usually small number of models employed, the ubiquity of the hypothesis/prediction/verification paradigm, and the local nature of most of the processing. Our work lies in the lower levels identified by the AI grouping, and could be regarded just as a system for deploying line finding techniques (cf. Yachida et al 1979). However, we have constructed an extensible system in which hypothesis/prediction/verification is clearly related to a hierarchical model description and the deployment of low level techniques. We suggest that this is a key area for image analysis, and one has been underemphasised in comparison to the representation of more articulate knowledge (such as 3D geometry or the layout of airports). It seems unlikely that any monolithic system will handle both these areas well.

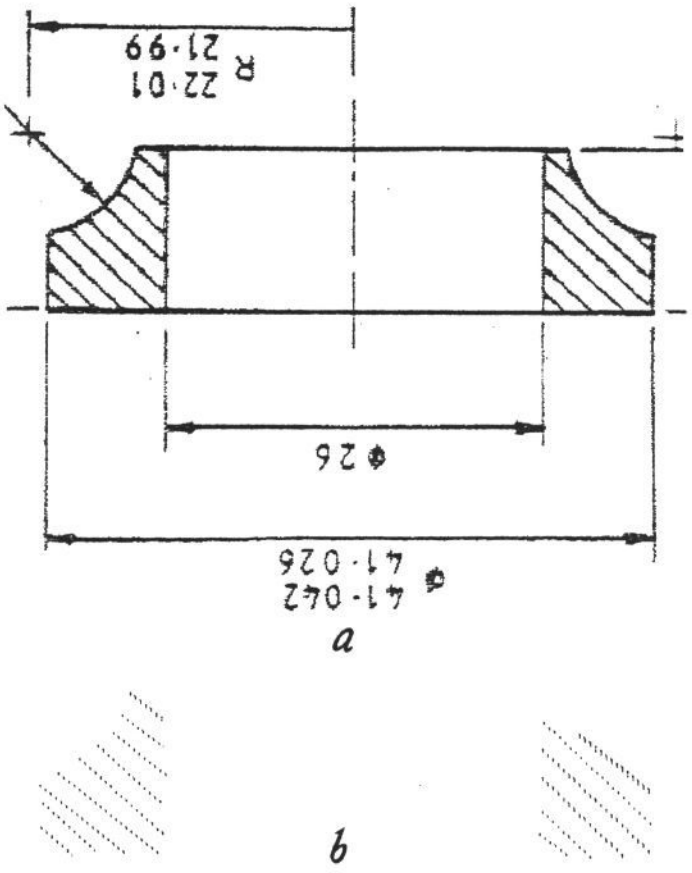


Figure 3. a) A thresholded drawing image. b) System output, showing solid crosshatching as finely pecked lines.

In the present work grammatical methods are seen as providing a layer between image processing and the higher level, more global processes needed to integrate pieces of drawing into a coherent whole. The interpretation of any given image will almost certainly require many calls to the parser. A similar approach is adopted in SPAM (McKeown et al 1985) and the later versions of MAPSEE (Mulder et al 1987). In both systems frame-like data structures, called schemata and functional areas respectively, are created and filled before some more global processes (constraint propagation in the case of MAPSEE, higher level rules in SPAM) resolve conflict between competing frames. Our objects are equally frame-like, but we lack the necessary higher level component. This extension will be the subject of future research.

The literature from the syntactic pattern recognition grouping describes various applications of grammars to drawing description, though fewer to drawing analysis. To deal with the continuous variables involved a conventional grammar may be enhanced with attributes which are used to enforce predicates controlling rule reduction (Miclet 1986). These attributes are passed as parameters, like ours describing context, but the predicates are evaluated quite separately from token extraction. This is more sound theoretically, but requires the development of special parsing machines and precompilation techniques. Where these techniques are best developed (MIRABELLE, Masini and Mohr 1983), they are applied to small, discrete systems.

Graph and web grammars (Shi and Fu 1983) are a natural format for drawing description, but analysis by parsing them is limited by the problem of erroneous data. Graph matching and rewriting has been demonstrated on small systems, but methods of extension are not evident. Henderson and Samal (1986) highlight the lack of tools for interactive grammar definition and for precompilation.

In comparison, our work is less concerned to describe a drawing than to handle a sequence of incoming data and trigger appropriate reductions. A correct (or error checking) parse at one attempt is not expected, though a highly tuned filtering of allowable structures is. Our token extraction method also seems appropriate: a straight line is a good building block for directly constructing many drawing entities. It represents a good compromise between the high level objects used for image modelling (eg. Shi and Fu 1983) for which extractors do not exist, and the low level ones used in chain code grammar (Pavlidis 1977) which imply a heavy low level processing load on the parser.

A feature common to both AI and syntactic pattern recognition approaches is a laying down of a priori rules to define the range of scenes considered. We feel that the determination of appropriate rules is more a matter for experiment than for legislation; the knowledge engineering component of image analysis is an important one. In this regard a transparent interpretation process is essential.

6 Conclusion

The use of a finite state machine compiled from an LR(1) grammar provides a powerful means for controlling image analysis processes to extract graphical entities from engineering drawings, though modifications and extensions to the basic machine are required to handle context dependency and two dimensional search.

7 References

1. **Binford, T.O.** "Survey of model-based image analysis systems" *Int. Jnl. of Robotics Research* Vol.1 No.1 pp 18-63 (1982).

2. **Black, W., T.P. Clement, J.F. Harris, B. Llewellyn and G. Preston** "A general purpose follower for line structured data" *Pattern Recognition* Vol. 14 pp 33-42 (1981).

3. **Bley, H.** "Segmentation and preprocessing of electrical schematics using picture graphs" *Computer Vision, Graphics and Image Processing* Vol. 28 pp 271-288 (1984).

4. **Bunke, H.** "Experience with several methods for the analysis of schematic diagrams" *Proc. 6th Int. Conf. on Pattern Recognition, IEEE Computer Society* pp 710-712 (1982).

5. **Cox, B.J.** *Object oriented programming* Addison-Wesley (1986).

6. **Johnson, S.C.** "Yacc - yet another compiler compiler", *Comp. Sci. Tech. Rep. No. 32* Bell Laboratories, Murray Hill, New Jersey (1975).

7. **Joseph, S.H.** "Automatic conversion of engineering drawings to CAD format" *Pattern Recognition* in press (1988).

8. **Haralick, R.M. and D. Queeney** "Understanding Engineering Drawings" *Computer Graphics and Image Processing* Vol. 20 pp 244-258 (1982).

9. **Henderson, T.C. and A. Samal** "Shape grammar compilers" *Pattern Recognition* Vol. 19 pp 279-288 (1986).

10. **Hofer-Alfeis, J.** "Automated conversion of existing mechanical engineering drawings to CAD data structures: state of the art" *CAPE '86: Conference on Computer Applications in Production and Engineering* Copenhagen (1986)

11. **Lin, W.C. and J.H. Pun** "Machine recognition and plotting of hand-sketched line figures" *IEEE Trans. Man, Systems and Cybernetics* Vol. 8 pp 52-57 (1978).

12. **Lin, X., S. Shimotsuji, M. Minoh and T. Sakai** "Efficient diagram understanding with characteristic pattern detection", *Computer Vision, Graphics and Image Processing* Vol. 30 pp 107-120 (1985).

13. **Mackworth, A.K.** "On reading sketch maps" *Proc. 5th IJCAI* pp 598-606 (1977).

14. **Masini, G. and R. Mohr** "MIRABELLE, a system for structural analysis of drawings" *Pattern Recognition* Vol. 16 pp 363-372 (1983).

15. **McKeown, D.M., W.A. Harvey and J. McDermott** "Rule-based interpretation of aerial imagery" *IEEE PAMI* (7) pp 570-585 (1985).

16. **Miclet, L.** *Structural methods in pattern recognition* North Oxford Academic (1986).

17. **Mohr, R.** "Precompilation of syntactical descriptions and knowledge directed analysis of patterns" *Pattern Recognition* Vol. 19 pp 255-266 (1986).

18. **Mulder, J.A., A.K. Mackworth and W.S. Havens** "Knowledge structuring and constraint satisfaction: the MAPSEE approach" *Tech. Rep. 87-21* Dept. of Computer Science, University of British Columbia, Vancouver, Canada (1987).

19. **Pavlidis, T.** *Structural pattern recognition* Springer-Verlag, Berlin (1977).

20. **Shi, Q.Y, and K.S. Fu** "Parsing and translation of attributed expansive graph languages for scene analysis" *IEEE PAMI* Vol. 5 pp 472-484 (1983).

21. **Smith, R.W.** "Computer processing of line images: a survey" *Pattern Recognition* Vol. 20 pp 7-15 (1987).

22. **Sugihara, K.** *Machine understanding of line drawings* MIT Press (1986).

23. **Tudhope, D.S. and J.V. Oldfield** "A high-level recogniser for schematic diagrams" *IEEE Computer Graphics and Applications* pp 33-40 (1983).

24. **Yachida, M., M.Ikeda and S.Tsuji** "A knowledge-directed line finder for analysis of complex scenes" *Proc. 6th IJCAI* pp 984-991 (1979).


```
(1) drawing : broken.line {drawing_instantiate()};
(2)         | outline {drawing_instantiate()};

(3) broken.line : line BREAK line  {broken.line_instantiate()};
(4)             | broken.line BREAK line {broken.line_extend()};

(5) outline : line CORNER line   {outline_instantiate()};
(6)         | outline CORNER line {outline_extend()};

(7) line : LINE {line_instantiate()};
```

Appendix 1. A small subset of the grammar rules. Terminals are in upper case, non-terminals in lower. The colon marks the boundary between left and right hand side of a syntactic reduction, a vertical line the logical or relation. Action code is enclosed by curly brackets in C++ style notation. The rule numbers referred to in the text are given in round brackets down the leftmost column.