Syntactic Analysis of Technical Drawing Dimensions

Suzanne COLLIN and Dominique COLNET
Centre de Recherche en Informatique de Nancy*
Campus Scientifique, Batiment LORIA,
Boîte Postale 239,
54506 Vandoeuvre-lès-Nancy Cedex
France
Tel. +33 83.59.20.67
Email: collin@loria.fr, colnet@loria.fr

November 16, 1993

Abstract

We present a syntactic approach to technical drawing dimensions analysis. A specific grammar is used to describe dimensions of drawings. This grammar can be graphically designed by combining different graphic primitives. The algorithm used for analysis can start at different points of the grammar. The analysis proceeds bottom-up and top-down according to previously obtained results.

1 Introduction – Related Works

The transfer of technical drawings from paper to CAD software data-bases and the construction of a 3D model of an object from several 2D views drawn on a draft, are topical problems. There are several different types of data in an industrial drawing, such as graphics, nomenclature and dimensioning. Dimensioning gives the measurement of the parts with more precision than direct scale measurements on the drawing. This article deals with the problem of dimension extraction and interpretation.

Figure 1 shows a part of a complex technical drawing. The problem of dimension extraction is to find every dimension on the drawing. Each dimension must be labeled according to its type: for example, angular dimension, radial dimension or longitudinal dimension. We intend also to have the coordinates of the dimensions included in the drawing, especially the positioning of the associated arrows and the dimensioning text.

The document is first digitized then vectorized; at the same time, connected components are extracted. Arrows are recognized using a template matching method on polygonal approximation segments, and text components are separated from graphic components according to the classic morphology-based analysis [Antoine 92], [Fletcher 88]. After this low-level step, the drawing becomes a set of graphic primitives: segments, arrows and text blocks. The major problem is to find an adequate model for the dimension.

In syntactic methods, the shape structure can be represented (recursively or otherwise) in terms of simpler patterns represented by graphs, trees or strings, using mathematical

*CRIN

Figure 1: *A part of a complex drawing with a lot of dashed lines and text often written on lines.*

models. A syntactic approach is a suitable representation when the patterns are well structured; this is not only the case with dimensions, but also with chemical structures [Feder 71], flowcharts [Bunke 92], and three-dimensional objects [Lin 84]. The definition of formal grammars adapted by [Fu 74] for pattern recognition, especially string grammars or attributed string grammars, [Tsai 80], [You 80], is not very well adapted for modelling two-dimensional patterns. Other grammars are more efficient: graph grammars [Claus 79], [Bunke 82], tree grammars [Agui 79], or plex-grammars [Fu 74], [Bunke 92]. These grammar parsers are sometimes difficult to implement; graph languages are more complex than string languages. Parsers have been proposed by [Fu 83], [Flasiński 89].

In this paper we use the plex-grammar formalism to define the dimension model (§ 5.1). The parser is able to work bottom-up and top-down, without heuristics (§ 6). The *a priori domain* knowledge is completely included in the terminal symbols and in the pattern assembly operators. This facility makes the parser very general, allowing easy grammar extensions. In addition, our parser is able to recognize partial dimensions due to vectorization distortions. This makes our algorithm interesting for applications in pattern recognition where parts of an input structure may be missing.

2 Model Presentation

A dimension is a collection of elementary primitives such as segments and circular arcs, text blocks and arrow symbols. Each of them is extracted at the low level stage of processing [Antoine 92], and is characterized by certain data: edge data, positioning of the primitives in the drawing, etc. In order to combine all these basic primitives for recognizing a dimension, we introduce the concept of *graphic primitives*, the *assembly operators* used, and finally the *grammar* itself.

Graphic Primitives (§ 3) are extracted at the low level stage. The drawing is first processed to find as many of them as possible. In the current implementation, primitives are segments, text and arrow heads.

Assembly Operators (§ 4) are used to combine graphic primitives one with another. All operators used are inspired by the work of [Shaw 69]. The operators specify how two particular points of two primitives (named *poles*) have to be put in coincidence.

The Dimension Grammar (§ 5) is composed of a set of rules enabling the description of higher level graphics. Defined at the beginning by D. Dori [Dori 88] such a grammar combines primitives according to their functional basis and morphology. A set of different rewriting rules, also called productions, gives the different possible definitions of the sub-shapes composing the dimension to be recognized. Finally, all different dimension classes can be described: longitudinal dimensions, angular dimensions, or circular dimensions. This grammar appears to be suited for dimension description: a few productions (only 8) given in the appendix is enough to describe the ISO standard. There are two major standards,ISO and ANSI; They are quite similar, but differ mainly by the position of the measurement text. For example, an ISO standard text is above the tail segment, while the ANSI standard text is by the side of the tail segment.

3 Graphic Primitives

Graphic primitives are extracted at the low level stage, by vectorization and segmentation techniques. At this time, a large number of graphic primitives are collected everywhere on the drawing. Different kinds of graphical primitives can be provided by the low level stage. We now have three kinds of graphical primitives: segments, arrows and text blocks [Collin 91].

A primitive is characterized by some appropriate numeric attributes and by a specific set of poles. The **numeric attributes** of a primitive include information about the localization in the drawing. The **set of poles** deals with connecting possibilities of the primitive.

For instance numeric attributes for a segment can be: the polar coordinates (ρ, θ) of the straight line defining the supporting axis (θ is the angle with the x axis, and ρ is given by $\rho = x sin\theta + y cos\theta$), the length of the segment, its thickness, etc. For a segment, the poles are two points: the extremities of the segment. These points can be used for further bindings with other primitives. Figure 2 shows an example of the segment primitive. Each pole is characteristic by coordinates that can be computed using the numeric attributes.

4 Assembly Operators

A pole can be "*visible*" or "*invisible*". A pole is a *visible* pole if a connection with other poles of shapes is still possible; it is an *invisible* pole if it is a point which will never be connected to other poles. The invisible poles keep track of the previous connections in a given shape. The criterion of visible or invisible poles for a primitive or shape depends on the definition rule. Figure 3 gives an example of primitives binding: the visible poles are large black dots, and invisible poles are large white dots.

In figure 9 the visible poles are labelled $p1$ and $p2$; the visible poles are large black dots, and invisible poles are large white dots.

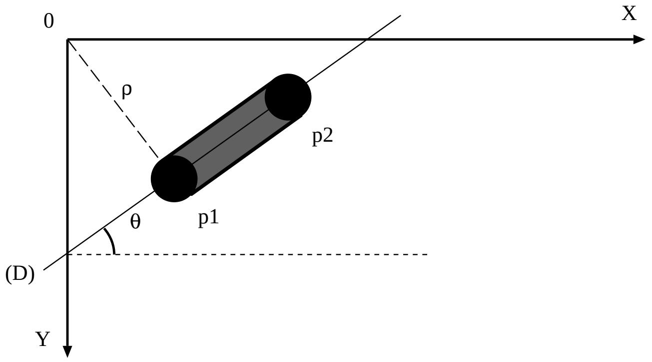


Figure 2: *The segment primitive. Numeric attributes are θ and ρ. Large black dots $p1$ and $p2$ are the two poles for further connections.*

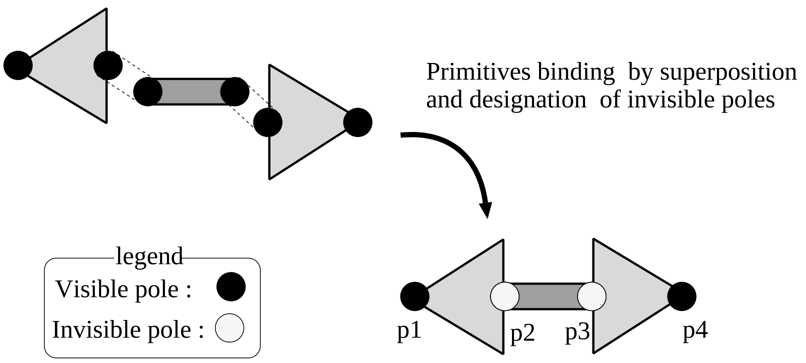


Figure 3: *Visible and invisible poles for a shape. Poles $p2$ and $p3$ are invisible poles. Only the visible poles $p1$ and $p4$ can be used for further connections.*

A **shape** is a collection of gathered primitives. In a dimension, the primitives are gathered according to the ISO or ANSI dimension normalization rules. We define two operators: the **pole superposition** operator and the **topographic** operator. The first collects together two shapes using a visible pole of each of them. The resulting pole is visible or not visible, according to the two associated shapes and the result shape.

The second operator associates two shapes according to a topographic relation between them; for instance, the relations *above*, *below*, *by_side_of*, etc. This relation is useful to describe the subshape composed with `text` *above* `segment`, or `text` *by_side_of* `segment`. The association *above(*`text, segment`*)* defines two visible poles, these belong to the segment, and two invisible poles, these belong to the text (cf. figure 6).

As we will see in § 5.2.1, the concatenation operator is defined without angular constraint on the associated shapes. This operator only associates two shapes by taking particular poles of each.

5 Dimension Grammar

In this paper we consider plex-grammars according to [Feder 71] as a special type of grammar formalism generating two-dimensional structures. The following basic concepts and definitions are included in this paper for the purpose of self-containedness. Definitions are the basis of the model concepts; They are general and defined for every plex-grammar including the concept of visible and invisible poles. Definitions are used to describe the parser § 6. Vocabulary and notations are those defined in [Fu 74] or [Feder 71].

5.1 Basic Concepts and Definitions

Definition of a Nape

Each shape can be visualized as having "attaching points" at which it may join or be associated with other forms. A general shape can be envisaged as having an arbitrary number n of attaching points for connection with other symbols. A shape with n attaching points is called an *N-Attaching Point Entity* or *nape.*

In our application, each *nape* is a shape composed only by its visible points numbered from 1 to n.

Definition of Plex-structure

A plex-structure is a triple $\Sigma = (\Psi, \Gamma, \Delta)$ with :

. $\Psi = (\psi_1, \psi_2, \ldots, \psi p)$ a list of p *napes*,

. $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_{p-1})$ the list of connections between **two** poles (or attaching points) of **two** *napes*, where :

$$\forall i = 1 \; to \; p-1, \quad \gamma_i = ((\psi_i, n_i^{i+1}) \; op \; (\psi_{i+1}, n_{i+1}^{i}))$$

n_i^{i+1} is the number of the visible pole in ψ_i matched with the visible pole number n_{i+1}^{i} in ψ_{i+1}, and *op* a concatenation operator defined on two *napes.*
The set Γ is called the set of *internal connections.*

. $\Delta = (\delta_1, \delta_2, \ldots, \delta_n)$ list of the n visible poles of the plex-structure.
Note : we have

$$\Delta \subseteq \bigcup_{i=1}^{i=p} \Delta_{\Psi_i}$$

where Δ_{Ψ_i} is the set of visible poles of the nape number i.

Definition of a Plex-grammar

A plex-grammar is a 4-tuple $G = (V_T, V_N, P, V_A)$, where

. V_T is a non-empty finite set of *napes*, called **terminal napes**. Each terminal nape has a non-empty set of visible poles.

. V_N is a non-empty finite set of *napes*, called **nonterminal napes**.
We have : $V_T \cap V_N = \oslash$ and $V = V_T \cup V_N$.

. P is a finite set of **productions** or rewriting rules.

. V_A where $V_A \subset V_N$, is a set of nonterminal *axioms*, i.e. nonterminals which describe a complete dimension. These particular nonterminals are not connected with any other shape, and have no visible pole.

Definition of a Production

A production is an entity denoted by

$$A \; \Delta_A \longrightarrow \Psi \; \Gamma \; \Delta$$

where

. (Ψ, Γ, Δ) is a plex-structure
. A is a nonterminal and Δ_A its visible poles.

The derivation of a context free plex-grammar proceeds by replacing the nonterminal nape in the left-hand side of a production with the right-hand side plex-structure.

The language $L(G)$ generated by a grammar G consists of sets of plex-structures each describing an individual subshape of the dimension. $L(G)$ is called a *plex-language*.

We can now apply these concepts to define the dimension grammar, i.e. to define V_T, V_N and P.

5.2 Definition of the Graphic Plex-grammar

Naturally, for each plex-grammar, a corresponding graphic representation exists. The two definitions, the formal one and the graphic one, are of course equivalent. In addition, we explain in § 5.2.2 how the graphic definition can be efficiently used by a *graphic dimensions editor*.

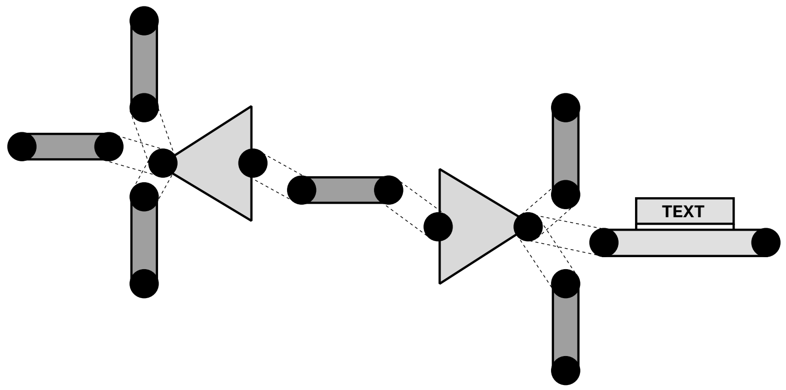


Figure 4: *Each primitive has a different "colour". There are 9 primitives: 6 segments, 2 arrows and 1 text block primitive.*

5.2.1 Terminal Symbols

The use of grammars in pattern recognition is based on the following idea: terminals of the grammar correspond to graphic primitives, or elementary pattern constituents, which can be directly extracted from an input image by means of suitable preprocessing such as segmentation methods. To provide an adequate description of the patterns, the primitives should contain information which is necessary for the specific application: size, location or semantic information.

In our case, we have three kinds of graphic primitives: segments, arrows and text blocks (cf. § 3). It is not a good idea to choose the segment structure as the only terminal symbol because there are too many segments in the drawing, and they are not characteristic of dimensions. Figure 4 shows a dimension split into segments, arrows and text blocks. With more complex primitives, simpler structural descriptions (i.e. simple grammars) of the pattern could be used. This tradeoff may become very important in the implementation of the recognition system. For instance, if we described the dimension only in terms of segments, arrows and text blocks, the structural description would be more complex than the case of using higher level primitives such as, for instance, the *TextBlokSegment* primitive (cf. figure 5). This one is *characteristic* of every ISO dimension[1]; it consists of a text block and the unique segment just under the text.

The definition of other more "complex" entities built from the three previous graphic primitives is possible because of the strong normalization of technical drawings. Figure 5 shows another split of dimension; two other entities appear as terminal symbols (three segments building a `T` and two vertical connected segments).

Finally, we define six possible terminal symbols (figure 6).

$$V_T = \{Segment,\ Arrow,\ TextBlockSegment,\ ThreePointsSegment,\ Corner,\ T\}$$

The choice of these terminal symbols, and the normalized rules for dimensions drawings avoid definitions of constraints such as a perpendicularity constraint, or angular constraint for instance. As a matter of fact, it is very unlikely that the dimension shown in figure 7 appears in a drawing. The lack of constraints enables recognition, without needing any threshold, the correctness of the dimension in figure 8.

[1] Some small modifications are too be made to consider the ANSI dimensions because in that case, text can be *by_side_of* a segment.

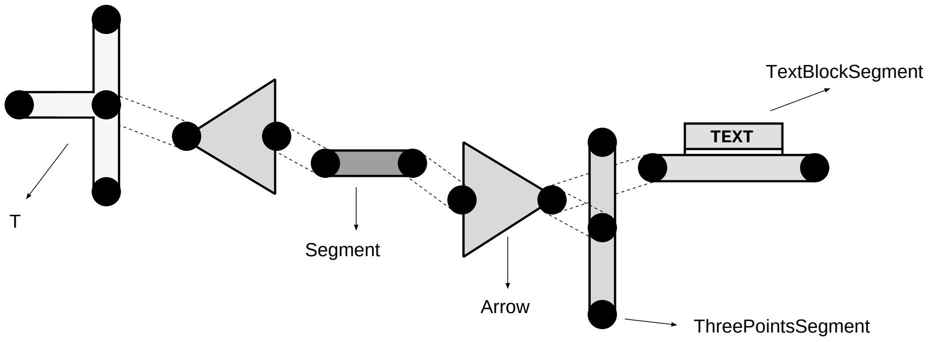


Figure 5: *Use of higher level terminals: for example, introduction of a* `T` *model in place of three segments (cf. figure 4).*

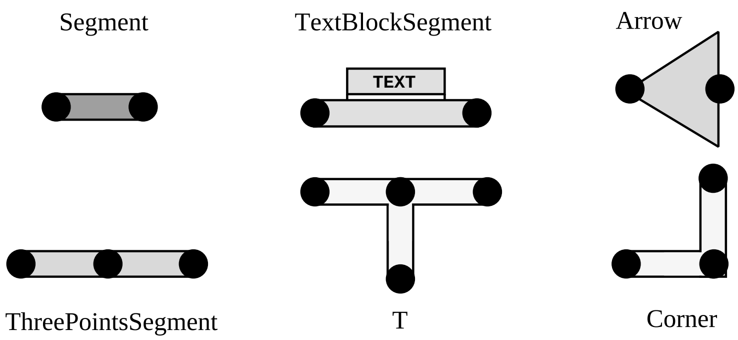


Figure 6: *Six possible terminals for the dimension grammar;* `Segment`, `TextBlockSegment` *including a topographical constraint,* `Arrow`, `ThreePointsSegment` *(two collinear connected segments)* `T` *(three segments with angular constraints) and* `Corner` *(two segments with right angle constraint).*

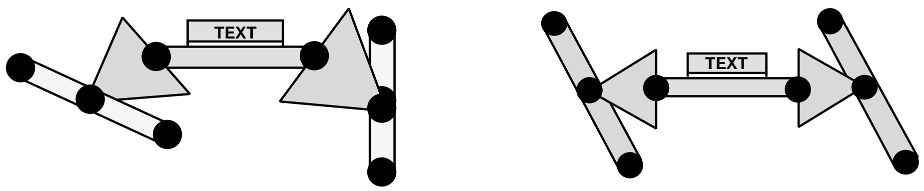


Figure 7: *This dimension is not correct; according to the normalization rules, it never appears in a drawing.*

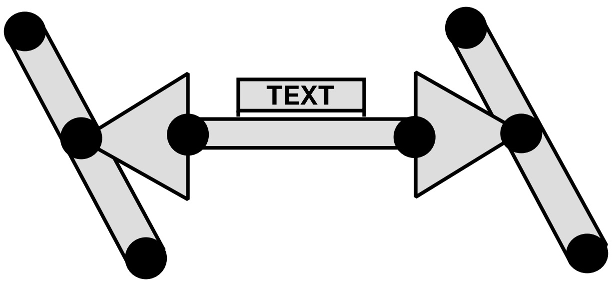


Figure 8: *Fortunately, this correct dimension is recognized.*

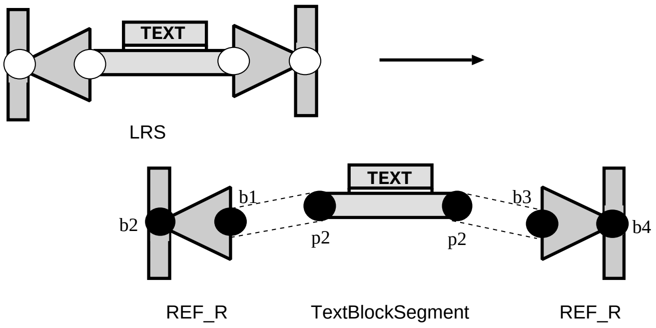


Figure 9: *Graphic description of a production. Visible poles are black dots, and invisible poles are white dots.*

In addition, if an angular constraint must be introduced, our strategy is to create a new specific terminal symbol including this constraint. Of course, this new subpattern should be again easily extracted by a low-level process, because terminal symbols are extracted only once at the low-level stage. This is the case with the terminal symbols `TextBlockSegment` and `Arrow`. Therefore, it is not necessary to define thresholds and heuristics in the parser itself. They are masked in the terminal symbol definition. We think that it is more efficient to look for complex primitives during the low level stage than to introduce thresholds and heuristics in the parser. The analysis is therefore very general and more reliable. This constitutes a particular aspect of our work; we show that it is possible to recognize dimensions without any specific constraints in the parsing process.

5.2.2 Graphical and Formal Description of a Production

A production (or a production rule) gives the different possible definitions of the subshapes composing the dimension to be recognized. A grammar rule can be applied any number of times, so it is possible to express in a very compact way basic structural characteristics.

For simplicity, terminal and nonterminal shapes are linked by only one pole of each giving as a result a unique nape. Also, each association of shapes is made with only two shapes.

Let us take an example (capital letters represent nonterminal symbols).

$$\begin{aligned} LRS() \;\longrightarrow\; & (REF\_R,\ TextBlockSegment,\ REF\_R) \\ & [((REF\_R,\ b1)\cdot(TextBlockSegment,\ p2)), \\ & \quad ((TextBlockSegment,\ p2)\cdot(REF\_R,\ b3))]() \end{aligned}$$

Figure 9 shows the *graphic* definition. According to the definition given in § 5.1 this production means that:

- $\Delta_A = \emptyset$, the resulting shape has no visible pole.
- $\Psi = \{REF\_R, TextBlockSegment,\ REF\_R\}$, represents the primitives of the LRS dimension.
- $\Gamma = [((REF\_R,\ b1)\cdot(TextBlockSegment,\ p2)),$ $((TextBlockSegment,\ p2)\cdot(REF\_R,\ b3))]$, is the list of linked poles. The first connection is made by attaching pole $b1$ of the first nape REF_R and by attaching pole $p2$ of $TextBlockSegment$. The other connections must be interpreted in the same way.

. $\Delta = \oslash$ since $\Delta_A = \oslash$ (no visible pole in the resulting shape).

The graphic description is much more expressive (figure 9) and carries the same information. The right hand side of a production is made by one or more connected entities (terminal or nonterminal). The left hand side describes the new entity that is to be defined; the visible and invisible poles are determined at this step. We give in the appendix an overview of the grammar. It is important to note that with eight rules we are able to describe most dimensions[2], except those built with arcs of circles. This shows the power of a syntactic description; the formalism of plex-grammar is well suited to dimension representation.

Another powerful two dimensional representation has been proposed: the formalism of graph grammar. In this formalism, the rewriting step operates on graphs, then the various relations between the components of a two (or n) dimensional structure are more effective. Different graph-grammars have been proposed and applied in pattern recognition [Claus 79] and especially to describe technical dimensions: D. Dori has proposed a *web-grammar*, [Dori 88], [Dori 92]. In our case, this formalism would be another powerful approach, but the parsing step for a graph language is more difficult than parsing for a string language. However, plex-grammars and graph-grammars are closely related [Fu 74]; sentences generated by a plex-grammar can be transformed into a directed graph by transforming primitives (or terminal symbols) to nodes and concatenation to labeled branches.

Each graphic production can easily be defined using a **graphic editor**. It is also possible to check productions (for instance arrow heads direction); A new production can be automatically transformed into an internal formalism recognizable by the parser. The user needs only know that a production is made with the permitted primitives (i.e. the terminal symbols) and contains no circularity in the association. This graphic editor is under implementation. With such a tool, it may be easy to define a plex-grammar for other domains (chemistry for example).

6 Dimension Interpretation by Syntactical Analysis

6.1 Description

In syntactic methods, the recognition procedure corresponds to parsing an unknown input pattern according to a given grammar. Our approach is based on R. Mohr's algorithm [Mohr 86] developed for an extension of the PDL language. Our parser is able to start with any primitive of a given technical drawing. Each terminal primitive can be a starter for the parser; however, the best choice is to take a primitive which guarantees the presence of a dimension (we call it a characteristic primitive), and which is always **well extracted**. The *TextBlockSegment* satisfies all these criterions and has been selected to be the starting point of the parser.

The parser is also able to recognize not only complete dimensions generated by the plex-grammar, but also partial ones. When performing the syntax analysis, the parser produces a complete syntactic description: a parse tree of the dimension.

Errors are neither incorporated in the model, nor in the parsing. Our strategy is to start by carrying out, syntax analysis without backtracking, even if some noise occurs at one coincidence pole; the analysis continues from the other poles, or stops if this is not possible. After this step, the objective is to look at the resulting extracted dimensions. As the tree contains information about them, such as their name, location in the drawing, coordinates of the missing connection poles, etc., it is possible to go back to low level processing to explore

[2] Longitudinal and diametric dimensions.

in more detail some parts of the image. If the resulting dimension is incomplete (i.e. it doesn't belong to the class of patterns described by the given grammar), and if the number of graphic primitives is not less than a fixed number, a second step is performed to correct the previous result. This number is the unique threshold included in the correction phase of the parser.

It is important to note that the parser is able to give good results without any heuristics (see the appendix). In addition, a new production can easily be introduced in the grammar, since the parser will not be modified.

6.2 Algorithms

The following algorithms are general but depend on the plex-grammar structure defined § 5. An algorithm able to start at any part of the pattern was developed by R. Mohr [Mohr 86]. It was designed for a language inspired by the work done by Shaw [Shaw 69]. Our algorithm is an extension of Mohr's parser. It is also able to start with any primitive of a given drawing, and proceeds alternatively bottom-up and top-down. It is extended to plex-grammars, i.e. not limited to shapes with only two particular connection poles (poles *head* and *tail*).

Our algorithm works as follow: let us suppose that a subshape F is recognized. Then we choose a production r such that $r : D \rightarrow \alpha_1 F \alpha_2$. This is made by the bottom-up procedure. If we have recognized the shapes α_1 and α_2, we therefore recognize the shape D.

The top-down procedure has to recognize the two shapes α_1 and α_2 . The analysis begins from the poles of F, extending the construction by the visible poles. The analysis is complete when we can apply a production $axiom \rightarrow B$, where B itself is completely recognized.

For example, let us consider the two following productions:

(1)	LRS	$\longrightarrow$	$REF\_R \cdot TextBlockSegment \cdot REF\_R$
(2)	REF_R	$\longrightarrow$	$ThreePointsSegment \cdot Arrow$

Suppose that the starting primitive is the shape `TextBlockSegment`; The first stage is to look for this primitive in the set of productions. In the example, the primitive is only in the production (1). The second stage starts with all productions found during the previous stage. We notice here that production (1) is used to start the analysis of the two shapes REF_R. Production (2) is selected by the top-down procedure; the model REF_R is matched with the graphic, using the fact that the `TextBlockSegment` is already localized. Moreover, when every symbol of the right-hand side of a production is localized, the corresponding nonterminal is also localized.

The inputs of the **bottom-up** procedure are :

. goal of the current analysis: $Goal$
. tree for the candidate subshape: a
. set of visible poles associated to the candidate subshape: P

The results are:

. recognized subshape tree: $TREE$
. shape poles set: $Pole$

The objective is to analyse a shape $Goal$ when a subshape with tree a, is candidate (cf. figure 10).

$Symb(a)$ returns the root of a, and $Pole \uplus p$ defines the appending of p at $Pole$ if and only if p is a visible pole.

The function f is defined as follow:
suppose B_i is a shape such that $\gamma_i = ((B_i, n_i^{i+1}) \cdot (B_{i+1}, n_{i+1}^i))$ (that means that B_i and B_{i+1} are connected by the poles n_i^{i+1} and n_{i+1}^i, cf. § 5.1), then $f(B_i, \Gamma) = n_i^{i+1}$, and $f(B_i, \Gamma)$ returns the geometric point from the visible pole of B_i, numbered n_i^{i+1}. The function f' returns the number of a pole.

```
procedure BOTTOM_UP (Goal, a, P, res : TREE, Pole)
 if Goal = Symb(a)
  then res ← (a, P)
  else
    choice of a rule r_p : D → Φ(C_1, ..., C_p) where ∃i, C_i = Symb(a)
    TREE ← a_i  /* a_i is the tree with shape C_i */
    for i = (i − 1) to 1 do
     TOP_DOWN (C_j, f(C_{j+1}, Γ_p), f'(C_j, f(C_{j+1}, Γ_p), Δ_p, res : a_j, P_j)
     TREE = TREE × a_j
     Pole = Pole ⊎ P_j
     /* P_j set of poles for C_j recognized */
    endFor
    for i = (i + 1) to p do
     TOP_DOWN (C_j, f(C_{j−1}, Γ_p), f'(C_j, f(C_{j−1}, Γ_p), Δ_p, res : a_j, P_j)
     TREE = TREE × a_j
     Pole = Pole ⊎ P_j
    endFor
    BOTTOM_UP (D, TREE, Pole, res : TREE, Pole)
 endIf
end
```

Figure 10: *Bottom-up analysis for multi-poles shapes.*

The first call is

$$\text{BOTTOM\_UP}(X,\ TextBlockSegment,\ liste\_pole\_visible(TextBlockSegment),\ \textbf{res} : TREE,\ Pole)$$

where X is the axiom and $TextBlockSegment$ the starting primitive. This is the most reliable graphical primitive.

The **top-down** procedure is given just below.

This procedure is looking for a shape F from a geometric point x which corresponds to the pole number k of F.

Its parameters are:

. the name of the subshape to be analyzed: F
. the starting geometric point: x

. the pole of F to link at x: k
. Δ_i and Γ_i are the notations defined in rule $r_i \ : \ A \ \Delta_i \ \longrightarrow \ \Psi_i \ \Gamma_i \ \Delta_i$.

The results are:

. a tree of the recognized shape: A
. a poles set of the recognized shape: $Pole$

```
procedure TOP_DOWN(F, x, k, Δi, res : A, Pole)
 if F ∈ T    /* F is a terminal, therefore a primitive */
  then if Pk(F) = x
   else
     A = A × F    /* × represents the "root" operation */
     Pole = Δi
   else error
  else
   choice of a rule ri : F → Φ (B1, ..., Bn)
   for i = 1 to n do
     TOP_DOWN(Bi, f(Bi, Γi), f'(Bi+1, f(Bi, Γi)), Δi, res : A, Pole)
   endFor
 endIf
end
```

Figure 11: *Top-down analysis for multi-poles shapes.*

The choice of a rule is a real problem when the grammar is made up of a lot of productions with *alternatives*, because the analysis can lead to an exponential time complexity. In this case, contextual information can be introduced to limit the possible choices. For instance, in the choice of a rule $D \ \rightarrow \ \lambda$ where C occurs in λ, it is obvious that a look at the primitives occurring in the *neighbourhood* of C may reduce the possible choices if the rule contains an alternative such as $D \ \rightarrow \ \lambda \mid \lambda'$.

This point is well discussed in [Mohr 86]; the definition of *context* of a primitive and the definition of the *neighbourhood* at a pole can reduce considerably the number of choices. This is very similar to what is done for LL(1) parsing by testing one primitive ahead.

However, in our case there is no alternative in the rules; the problem of the multiple choices doesn't really occur, although there is no guarantee that every shape has a different context. In our approach, context and neighbourhood processing is available if the choice of a rule at a given time must be made between a lot of productions (three or four possibilities). Our strategy is based on the analysis of all the proposed possible choices, and the execution of all current analyses. Parsing then leads either to a success, or to a failure; in the latter case, the analysis is abandoned at the failure point. Nevertheless we are guaranteed that each success case is explored.

7 Conclusion

We have developed a system able to recognize dimensions in technical drawings. Examples of successful recognition are given in the following appendix. No character recognition has

Figure 12: *Original drawing and split drawing.*

been used during the different steps of analysis. The error correcting mechanism has not yet been implemented. It is processed after parsing, only on incomplete dimensions. During parsing, a syntactical tree is built, keeping the maximum of information about the analysis: current rule, missing primitive and its location on the drawing, etc. The parser contains no heuristics; the algorithm is very general and therefore easily allows grammar extensions. The choice of the starting primitive is not fixed, and it is possible to choose either arrow, or text block, or any other terminal without changing the parser strategy.

This recognition system is purely syntactic because of the nice structural description of dimensions. It shows that grammar formalism is a powerful and compact way to describe shapes that are very well structured.

This work on plex-grammars leads us to think about the interpretation of a whole drawing. An extension of such syntactic techniques would be to define a plex-grammar to link blocks not by points but by *segments*; figure 12 shows a mechanical piece split into blocks. We think it is possible to recognize in such drawings, first some fundamental objects (such as a shaft), and then the whole drawing. This is our next objective.

References

[Agui 79] T. Agui, M. Nakajima, T.K. Kim and E.T. Takahashi. A Method of Recognition and Representation of Korean Characters by Tree Grammars. *IEEE Transactions on PAMI*, 1(3):245–251, 1979.

[Antoine 92] D. Antoine, S. Collin and K. Tombre. Analysis of Technical Documents: The REDRAW System. In H.S. Baird, H. Bunke and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 385–402. Springer-Verlag, Berlin/Heidelberg, 1992.

[Bunke 82] H. Bunke. Attributed Programmed Graph and Application to Diagram Interpretation. *IEEE Transactions on PAMI*, 6(4):574–582, 1982.

[Bunke 92] H. Bunke and B. Haller. Syntactic Analysis of Context-Free Plex Languages for Pattern Recognition. In H.S. Baird, H. Bunke and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 500–519. Springer-Verlag, Heidelberg, 1992.

[Claus 79] V. Claus, H. Ehrig and G. Rozenberg, editors. *Graph-Grammars and Their Applications to Computer Science and Biology.* Lecture Notes in Computer Science. Springer-Verlag, 1979.

[Collin 91] S. Collin and P. Vaxivière. Recognition and Use of Dimensioning in Digitized Industrial Drawings. In *Proceedings of First International Conference on Document Analysis, Saint-Malo, France*, volume 1, pages 161–169, 1991.

[Collin 92] S. Collin. *Interprétation de la cotation des dessins techniques par analyse syntaxique.* Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, Janvier 1992.

[Dori 88] D. Dori and A. Pnueli. The Grammar of Dimensions in Machine Drawings. *Computer Vision, Graphics and Image Processing*, 42:1–18, 1988.

[Dori 92] D. Dori. Self Structural Syntax-Directed Pattern Recognition of Dimensioning Components in Engineering Drawings. In H.S. Baird, H. Bunke and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 359–384. Springer-Verlag, Heidelberg, 1992.

[Feder 71] J. Feder. Plex Languages. *Information Science*, 3:225–241, 1971.

[Flasiński 89] M. Flasiński. Characteristics of edNLC-Graph Grammar for Syntactic Pattern Recognition. *Computer Vision, Graphics and Image Processing*, 47:1–21, 1989.

[Fletcher 88] L.A. Fletcher and R. Kasturi. A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. *IEEE Transactions on PAMI*, 10(6):910–918, 1988.

[Fu 74] K.S. Fu. *Syntactic Methods in Pattern Recognition.* Academic Press, New York, 1974.

[Fu 83] K.S. Fu and Q.-Y. Shi. Parsing and Translation of Attributed Expensive Graph Languages for Scene Analysis. *IEEE Transactions on PAMI*, 5:472–485, 1983.

[Lin 84] W.C. Lin and K.S. Fu. A Syntactic Approach to 3-D Object Representation. *IEEE Transactions on PAMI*, 6(3):351–362, 1984.

[Mohr 86] R. Mohr. Precompilation of Syntactical Descriptions and Knowledge Directed Analysis of Patterns. *Pattern Recognition*, 19(4):255–266, 1986.

[Shaw 69] A.C. Shaw. A Formal Picture Description Scheme as a Basis for Picture Processing Systems. *Information and Control*, 14(1):9–52, 1969.

[Tsai 80] W.H. Tsai and K.S. Fu. Attributed Grammar: A Tool for Combining Syntactic and Statistical Approaches to Pattern Recognition. *IEEE Transactions on PAMI*, 10(12):873–885, 1980.

[You 80] K.C. You and K.S. Fu. Distorted Shape Recognition Using Attributed Grammars and Error-Correcting Techniques. *Computer Vision, Graphics and Image Processing*, 13:1–16, 1980.

Appendix

Experimental Results

The grammar and the parser are implemented in Smalltalk-80. The use of an object oriented language enables us to write very short "methods", and the dynamic binding mechanism avoids writing lots of tests. It is entirely based on a message sending mechanism: the analysis is led by messages protocols enabling communication between the instances of classes [Collin 92]. Each graphic terminal, nonterminal and production is represented by classes, hierarchically organized. Bottom-up and top-down procedures are managed by a *question/reply* protocol.

We give here some experimental results. Figure 13 (a) shows the original raster image (scanner at 300dpi) the second (figure 13 (b)) shows the set of primitives: segments, arrows and text blocks. In figure 14 (a) we see complete dimensions, and in figure 14 (b) the set of incomplete dimensions (consequences of missing primitives).

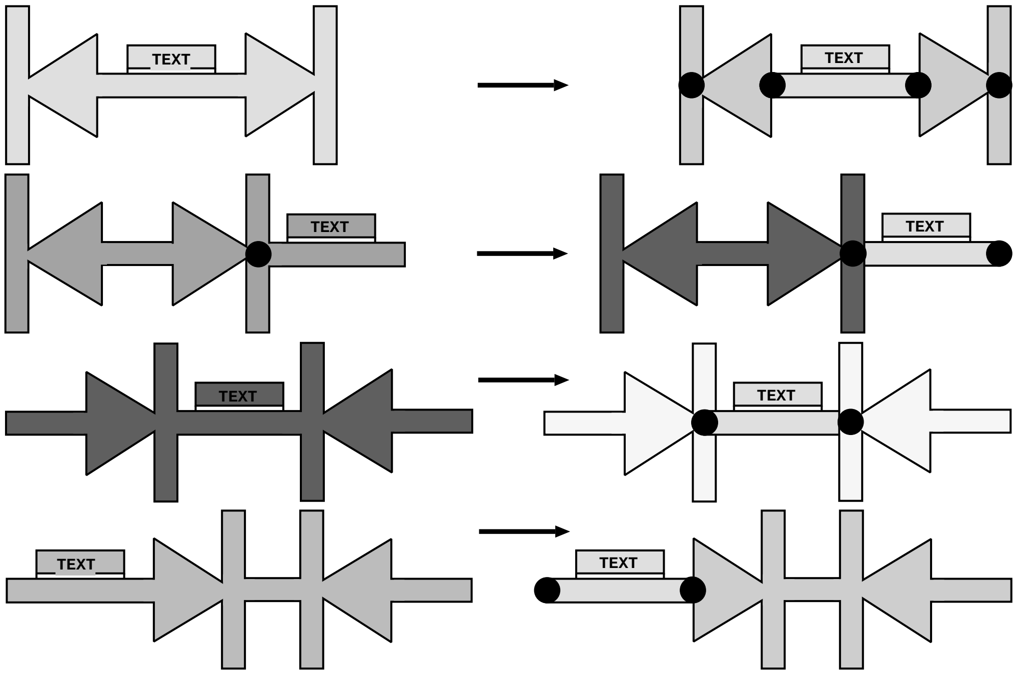
The run time of the parsing process is nearly 3 seconds on a SUN 3 workstation and what is needed for extraction of primitives (segments, arrows and text blocks) is about 20 seconds for the example shown.

Our system is tested on 10 drawings without any errors for complete recognized dimensions; each complete recognized dimension is **really** a complete one. If a dimension is not fully recognized: then it is not complete: i.e. an arrow head or a segment is missing. This is certainly due to our choice of vectorization process. We are currently working to improve our results by studying incomplete dimensions and trying to complete them. Therefore, it becomes possible to return back to the original image and complete the concerned dimension by bringing about an improved local vectorization.

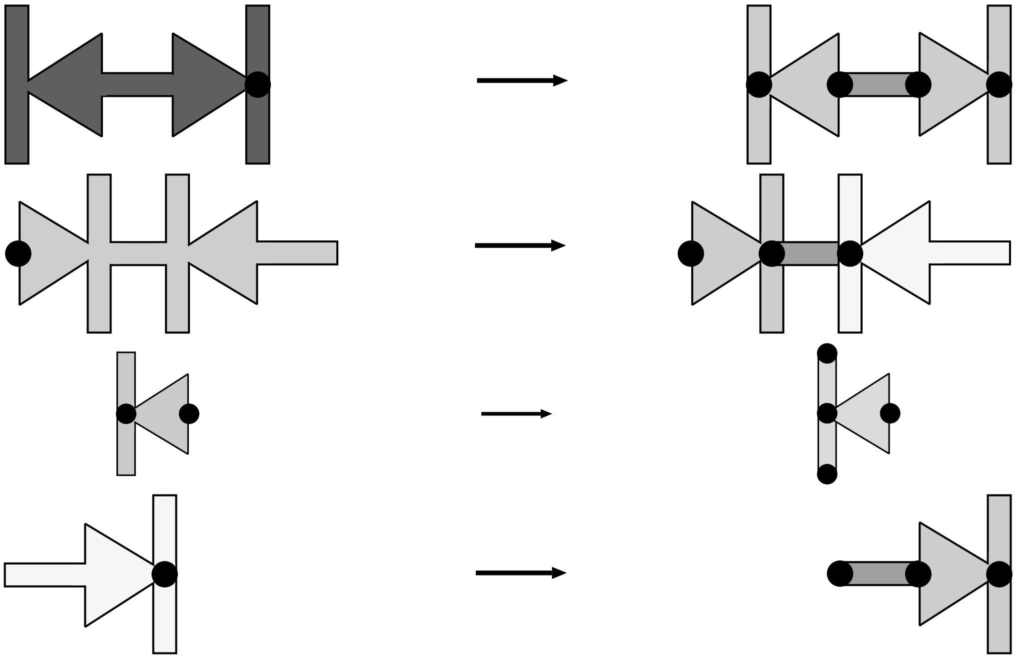
Dimension Graphic Grammar

We give in this section the graphic grammar able to recognize longitudinal and diametric dimensions. Productions one to four are the axioms: they represent complete dimensions. Only visible poles are shown below: they are shown as black dots. The last line represent the four terminal symbols used in our implementation.

Axioms:



Other productions:



Primitives:

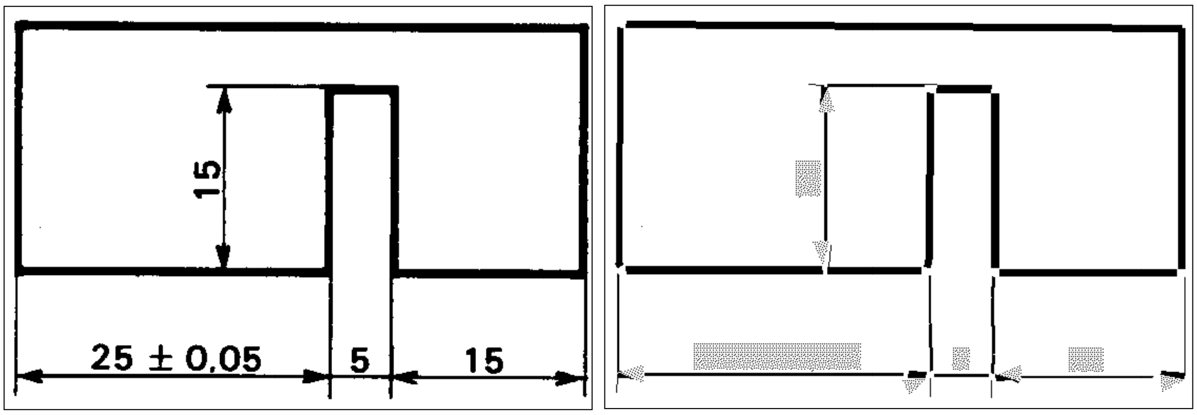


Figure 13: *(a): original image* *(b): extracted primitives (segments, arrows, text blocks)*
-

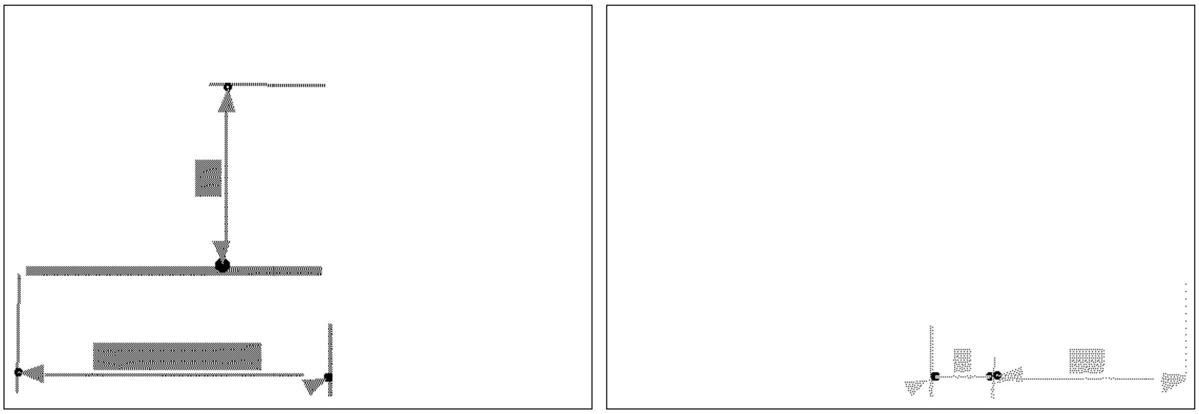


Figure 14: *(a): complete dimensions* *(b): incomplete dimensions*