a string, not containing end-of-lines or end-of-file markers, followed by the end-of-line character. A *file* is a sequence of lines followed by the pseudo-line containing exactly the single character \e.

We deal with several kinds of sequences. We adopt the convention that any single object is also a sequence of length one consisting of that object. The concatenations of strings, segment sequences and token sequences are denoted by ¦, ! and $\circ$ respectively. Note that sequences of lines, or of segments are also strings. We use regular expression notation when requiring sequences of a certain pattern. Thus, $x ** k$ stands for the sequence x repeated k times, and $x*$ stands for $x ** k$, for some $k \geqslant 0$. Unless explicitly stated otherwise, by string we mean a string of characters free of \*e*. We show strings enclosed in double-quotes. A string x is a prefix of z if $z = x ¦ y$ for some y; x is a suffix of z if $z = y ¦ x$, for some y. The words prefix and suffix have analogous meaning when referring to other kinds of sequences. Empty string, token sequence and segment sequences are denoted respectively by "", 00 and 0ss.

The specifications require many predicates and mathematical functions. We use names with upper-case letters in them for these. In the definitions read '::=' as 'is defined as'.

4. HIGH LEVEL SPECIFICATIONS

In this section we specify the layout of programs using the Extended BNF grammar[7] of Pascal. As the lexical structure of Pascal is left undefined there we expect the reader to use his own intuitive understanding of how a string is mapped to a token sequence, for the time being. We also ignore, until the next section, the presence of comments, as does the above syntax definition. We also make minor changes to the grammar. For instance, all occurrences of terminal strings are replaced by non-terminals whose names are composed of the letters *nt* followed by the name of the token (in upper case). Thus the nonterminal *nt*REPEAT produces w ¦ "*repeat*", where w stands for a (possibly empty) white space

Given a string s with no white space suffix, $s = s_1 ¦ s_2 ¦ \ldots ¦ s_k$, and the corresponding production rule $n = n_1\, n_2 \ldots n_k$, such that $n \rightarrow^* s$, $n_j \rightarrow^* s_j$, we assume that the s_j do not have white space suffix. However, the s_j may have a prefix white space. This is significant as the white space prefix of each line is, so to speak, all that matters.

We say that a given string s corresponding to a non-terminal n is 'properly laid out' starting at margin m if PLOT(n, s, m) = **true**. The definition of PLOT is given compactly in a syntax-directed way in Figure 2. Each production acts as a template for a conjunction of NEWL and PLOT predicates, which are defined below; substituting actual strings for the non-terminals gives a logical conjunction which can then be evaluated. Each line in the diagram contains one terminal (which we show by the appropriate token) or one non-terminal (and possibly a metabracket) whose indentation from the reference vertical gives the 'ruling margin' increment for it. The presence of a NEWL predicate is indicated by a \n character to the left of the reference vertical.

To conserve space, we have omitted from Figure 2 all productions whose specifications are of the form