Recall that we replaced terminal symbols appearing in the right-hand side of productions by their token names and considered the latter as non-terminals. Thus any zi that can be generated in one application can contain only one token and INDENT(zi) contains no blank space in front of this token; thus PLOT(t, $\backslash n \mid$INDENT(zi), 0) holds for some appropriate token t.

For the inductive step, note that $zo =$ INDENT(zi) can be divided such that $zo = zo_1 \mid zo_2 \mid \ldots \mid zo_k$, no zo_j contains white space suffix, and $N_j \rightarrow^* zo_j$. Further note that the zi_j, zu_j and zo_j are all lexically equivalent. Since $zu_j =$ INDENT(zi_j), zo_j possibly differs from zu_j only in the width of the margin of each line and by containing an extra $\backslash n$ in the prefix of zo. The rest of the proof of this step follows from these observations and is simple but tedious requiring case analyses for each non-terminal N of the grammar. Here we present two such cases—one for the repeat statement and another for the procedure declaration.

Case N = repeat statement

Clearly TKNSEQ(zo_1) = REPEAT and TKNSEQ(zo_{k-1}) = UNTIL. Also, for $1 < j < k-1$, zo_j must equal $c \mid (zu_j$ with the margin of each line of zu_j increased by UOI blanks). Here the string c is either empty, or is $\backslash n$ depending on the segment sequence SEGSEQ(zi). If a segment boundary fell between zi_{j-1} and zi_j, and if zi_{j-1} did not end with a $\backslash n$, then $c = \backslash n$, else $c =$ " ". From the definition of SEGSEQ, it follows that a segment boundary falls between z_{j-1} and zi_j either because zi_{j-1} terminated in a $\backslash n$, or in a token from LC followed by (portions of) comments, or because zi_j begins a token from LO. Since LO and LC were chosen so as to make NEWL predicates true, PLOT(*repeat statement*, $\backslash n \mid zi$, 0) must be true.

Case N = procedure declaration

We shall make further assumptions below for the sake of simplicity in this illustration. We have that TKNSEQ(zo_1) = PROCEDURE, TKNSEQ(zo_2) = ORDINARY (the corresponding word being the name of the procedure), TKNSEQ(zo_3) = SEMICOLON, assuming that the procedure heading has no parameters, and TKNSEQ(zo_k) = END. Further assuming that the procedure has only variable declaration part, we have TKNSEQ(zo_4) = VAR. Let $zo_5, \ldots, zo_{v-1}$ correspond to this declaration such that TKNSEQ(zo_v) = SEMICOLON, TKNSEQ(zo_{v+1}) = BEGIN. Clearly then, $zo_{v+2} \ldots, zo_{k-1}$ correspond to the code body of the procedure. Note that the value of NMG( ) will be 2*UOI starting from zo_5 until zo_v, both inclusive. After zo_{v+1} it becomes UOI and remains at least UOI until zo_k. As in the previous case, we see that the code body and the variable declaration and hence the procedure declaration thus meet the high level specifications.

7. CONCLUDING REMARKS

This section contains some remarks based on personal experience with this case study in specifying the behaviour of a medium sized program. I wrote the first version of an indenting program in late 1978 mainly as a reaction to the very long, slow and often clumsy indenting programs that were known to me at that time. A year later, I needed a class-room example of a real life program whose specification and proof are given sufficiently rigorously but with as little formalism as possible.