A syntax for ALGOL input/output formats

By W. A. Zaremba*

The desirability of simple and yet comprehensive input/output formats is postulated. A possible grammar for ALGOL formats is given and shown to be of a simple precedence type. A reduced precedence matrix has been included and the meaning of major constructs explained and illustrated using the regular expression language. The proposal is contrasted with the relevant part of a report by Knuth (1964) on ALGOL input/output conventions, and various trade-offs are briefly discussed.

(Received January 1969)

Basic (primitive) procedures in terms of which the input and output of ALGOL programs can be described have been defined in the report of Working Group 2.1 of IFIP (1964). In any practical application these procedures have to be augmented by a comprehensive set of other conventions and one such set has been defined and published as a report of a Committee of the Association for Computing Machinery (Knuth, 1964). Included in it are the proposals for formats, input/output procedures and control variables. It would be rather hard to improve on the scope of those proposals, in this article therefore we want to examine a much narrower field of format specifications only, and to exclude all discussions of input/output procedures, etc. The motivation for this work is twofold:

(*a*) There is a lot to be said for the simplicity of FORTRAN type format in which a prefix letter determines the type of conversion and the digits denote the field width.

(*b*) The full generality of formats defined by Knuth (1964) is unlikely to be required in practice, while some rather important cases of output such as the replacement of leading zeros by asterisks (for cheque protection, etc.) cannot be specified at all.

The object of this article then is to describe a set of alternative standard conventions for input/output formats. In particular, the syntax will allow for both a full pattern specification by picture as in Knuth (1964), and also a very compact FORTRAN-like representation, if so desired. Further, because of the use of a so-called line-up format as separator for the editing formats, a one-to-one picture pattern of entire line may be possible in many cases. The syntax given in the next section is unambiguous and follows the method of Wirth (1966). It was possible to make it a simple precedence type as well, thus its parsing is immediately available using the well-known standard algorithm. The full specification is rather lengthy with 166 productions rather than just 81 of Knuth (1964), this however is forced by the requirement of non-empty right parts in all productions and a 'linear' arrangement with just two terms in each definition (except in the last production No. 166).

We shall now informally describe the type of coding that might be desirable in specifying the output. The point of departure is a FORTRAN format like, say, F6.2. It would be natural to interpret this (unlike in FORTRAN) as implying a fixed point conversion with six fields ahead and two behind a decimal point. If we now wanted to separate the thousands to make for easier reading, we would wish to write something like F3S3.2 where S would indicate a blank field. A moment's reflection shows that there is really no need to distinguish the types **integer** and **real** by means of different formats, thus the next step is to adopt the approach of Knuth (1964) and to encode both fields by D; hence D3 indicates an **integer** field while D3.2 denotes a **real** field. For further generality we may include the 'picture formats' by allowing the repetition of code letters, thus D3 ≡ DDD and D3.2 ≡ DDD.DD, and adding the possibility of octal or binary conversions by the use of some obvious codes (say Q and B respectively) the system can be made quite powerful. (Note that the system described by Knuth (1964) could also be so extended, in principle though at the cost of adding more code letters.)

The standard interpretation of format should be the one most frequently required: i.e. a number after editing would be aligned at the decimal point and then printed in its field with all the leading zeros suppressed and the sign, '—' for negative and a blank for positive, in the rightmost suppressed digit position. Since for a zero value output nothing at all would then appear—so in order to force a visible entry we would like to indicate by means of some obvious code (say Z for 'zero') that starting with *that* position zeros are to be printed whether significant or not. Thus, for instance, D3S3 with a value of zero would give an all blank field of 7 columns but DZ2S3 would produce '00 000'.

Introducing now a special code, say F for 'fill-up' followed by * we could indicate that the non-significant zeros rather than being suppressed, assuming no Z code in the format, are to be instead replaced by the asterisks, and in general, by allowing for other special characters to follow the F we would trigger any special editing requirements left unspecified in the current work (e.g. F+ might print '+' or '—' rather than a blank or a '—', etc.).

For the output of character strings the A-formats can be introduced in a standard manner and with the same type of replication and internal separations, or even insertion literals, as for the numbers.

* *Standard Oil Co. of Calif. WOI, La Habra, California*

Lastly, by allowing the formats to be nested and the repetitions to be specifiable by remote elements through a code letter R (say), we obtain the complete system.

Syntax and precedence matrix

The syntax is presented here in a way similar to that used by Wirth (1966). The exact rules are:

1. All basic and non-basic symbols are written as contiguous strings of characters without embedded blanks.
2. One or more blanks must separate the symbols on the right side of productions, i.e. the blanks act there as concatenation symbols.
3. Successive productions with the same left part are listed below one another, i.e. a transfer to the next line acts as a metalinguistic 'or' and is equivalent to the | of ALGOL report.

For ease of machine representation some liberties were taken with the naming of basic symbols, e.g. LQ stands for 'left string quote', STRN for 'string', UINT for 'unsigned integer', 10 for $_{10}$ etc. The non-basic symbols have been assigned coded names only since most of them are forced by the precedence requirements and do not, of themselves, denote any conceptual objects. Those few that do will be assigned special names in the next section of the article (see: 'Interpretation and examples'). The precedence matrix (**Fig. 1**) following the syntax specification (**Table 1**) has been reduced by collapsing all identical rows (or columns) to list them just once.

More formally, if the range of bound variables is 1, . . . , n and K is any one largest set of indices of dissimilar columns of the precedence matrix M, we have the predicate *colset* (M, K) indirectly defining the set K:

$$(k1)(k2)\{k1 \in K \wedge k2 \in K \wedge k1 \neq k2 \rightarrow (\exists r)(M_{r,k1} \neq M_{r,k2})\}$$
$$\wedge (k1)\{k1 \notin K \rightarrow (\exists k2)(k2 \in K \wedge (r)(M_{r,k1} \equiv M_{r,k2}))\}$$
$$\equiv colset(M, K)$$

Similar expressions can be written for the rows of the $n \times n$ precedence matrix M. This procedure has decreased the size of matrix from 99 × 99 to just 85 × 35, i.e. to about 30·4% of the original size. Since each element can be coded 0, 1, 2 or 3 a natural idea would be to use just two bits for its representation and to pack the entire matrix into only about 186:32-bit words, though of course because of multiple column (row) headings the indexing gets more complex.

To save space and to get by using only the standard character set, the column headings in Fig. 1 read vertically down. Those reading as integers followed by ':' are to be correlated to entries in all the rows bearing these same integers, and denote the 'multiple' columns. For instance: 2: read vertically stands for the set ($C3$ $C4$) etc. Where a number followed by ':' appears in a row heading it applies to all symbols that follow behind until possibly another such number. Thus for example '6:$H1$ 9:LQ' means that $H1$ is included in the column headed by 6: (read vertically), while LQ is in the column under the heading 9:.

The syntax specification and the precedence matrix follow. Note that there exist no precedence functions for this system.

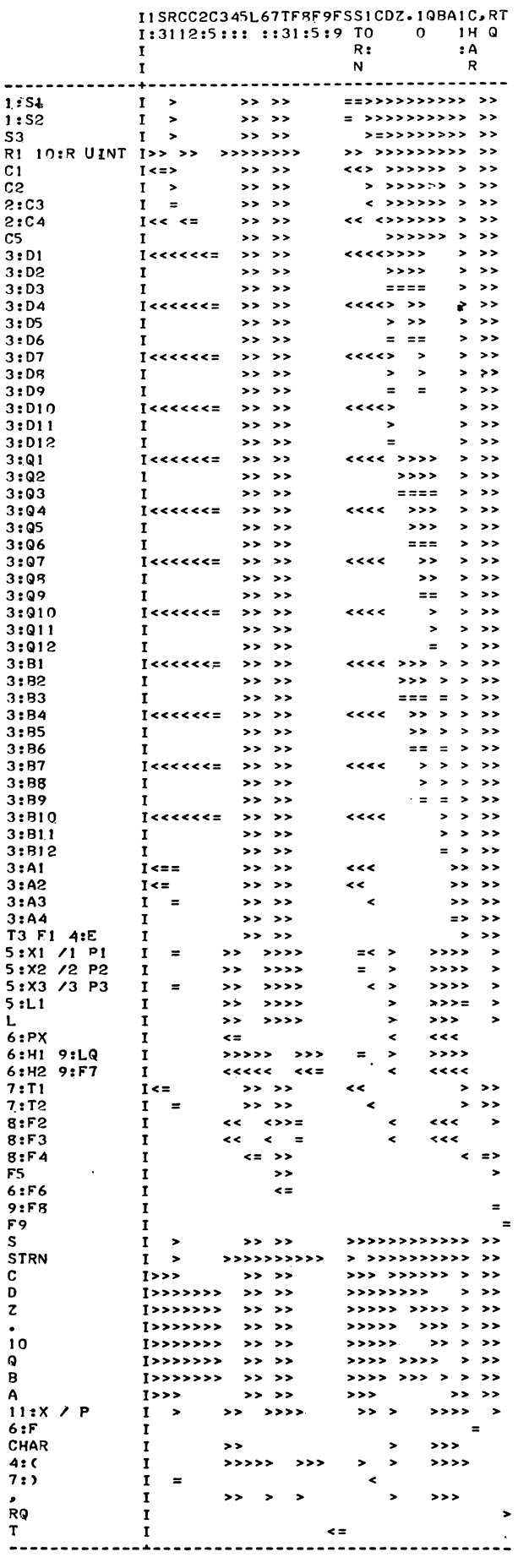

Fig. 1. Reduced precedence matrix for the format syntax

Interpretation and examples

The syntax given in the last section defines an unambiguous language, but it gives little insight as to legal constructions; in the discussion to follow we shall, therefore, use regular expressions for that purpose. These can also serve as another definition, especially if a recognizer be available. All regular sets have been defined so that they contain no empty strings, and this eases their recognition. In the terminology of Brzozowski (1964) we have:

$$\delta(G) = \phi \text{ for all } G.$$

In the current section we shall use the same conventions as before with the addition of scope brackets and the iteration operator, respectively: { } and *. The concatenation of regular sets is denoted by separating their

Table 1

Precedence syntax for I/O formats

```
 1: S1 → S
 2:    → S1 S
 3:    → S2 S

 4: S2 → STRN
 5:    → S1 STRN

 6: S3 → S1
 7:    → S2

 8: R1 → UINT
 9:    → R

10: C1 → C
11:    → S3 C

12: C2 → S3
13:    → C1
14:    → C1 S3

15: C3 → C2
16:    → C4 C2

17: C4 → R1
18:    → C3 R1

19: C5 → C3
20:    → C4

21: D1 → D
22:    → D3 D

23: D2 → D1 C5

24: D3 → D1
25:    → D2

26: D4 → D3 Z
27:    → D6 D

28: D5 → D4 C5

29: D6 → D4
30:    → D5

31: D7 → D3 .
32:    → D6 .
33:    → D9 D

34: D8 → D7 C5

35: D9 → D7
36:    → D8

37: D10 → D3 10
38:     → D6 10
39:     → D9 10
40:     → D12 D

41: D11 → D10 C5

42: D12 → D10
43:     → D11

PRØDUCTIØNS 44–66
SIMILAR TØ 21–43
BUT WITH LETTER "Q"
REPLACING "D".

PRØDUCTIØNS 67–89
SIMILAR TØ 21–43
BUT WITH LETTER "B"
REPLACING "D".

90: A1 → A
91:    → A4 A

92: A2 → A1 R1
93:    → A3 R1

94: A3 → A1 S3
95:    → A2 S3

96: A4 → A1
97:    → A2
98:    → A3

99: E → D3
100:  → D6
101:  → D9
102:  → D12
103:  → Q3
104:  → Q6
105:  → Q9
106:  → Q12
107:  → B3
108:  → B6
109:  → B9
110:  → B12
111:  → A4

112: X1 → X
113:    → L1 X

114: X2 → X1 R1
115:    → X3 R1

116: X3 → X1 STRN
117:    → X2 STRN

118: /1 → /
119:    → L1 /

120: /2 → /1 R1
121:    → /3 R1

122: /3 → /1 STRN
123:    → /2 STRN

124: P1 → P
125:    → L1 P

126: P2 → P1 R1
127:    → P3 R1

128: P3 → P1 STRN
129:    → P2 STRN

130: L1 → X1
131:    → X2
132:    → X3
133:    → /1
134:    → /2
135:    → /3
136:    → P1
137:    → P2
138:    → P3

139: L → L1

140: PX → F CHAR

141: H1 → (
142:    → PX (

143: H2 → H1
144:    → H1 STRN

145: T1 → ) R1
146:    → T2 R1

147: T2 → T1 S3

148: T3 → T1
149:    → T2

150: F1 → E
151:    → PX E
152:    → F6 T3

153: F2 → L
154:    → F4 L

155: F3 → F4 ,

156: F4 → F1
157:    → F2 F1
158:    → F3 F1

159: F5 → F2
160:    → F4

161: F6 → H2 F5

162: F7 → LQ
163:    → LQ STRN

164: F8 → F7 F5

165: F9 → F8 RQ

166: FT → T F9 T
```

names by one or more blanks, ϕ is an empty set, while | now denotes the set union. We shall again proceed from the constituents to the final expressions and consider first and foremost the action during output.

Insertion: S3→S S* | S* STRN {S S* STRN}* S*

Examples: S
SS'CR'S
'KG'S'M', etc.

This is an alternating sequence (at least one element) of subsequences of code S—each standing for a blank insertion, and/or strings—which are moved unchanged to the output. Insertions are intended for use *within* the subfields corresponding to the *same* edited item. (The spaces outside or between the items are to be specified by code X.) If a non-significant zero is suppressed to the left of S, that S-position will be replaced by a fill-up character (if within the scope of one). Note that S code cannot be replicated, i.e. S4 *does not* imply the expression SSSS, hence this code should only be used to obtain a reasonable amount of subfield separation.

Replicator: R1→UINT | R

Examples: 3
R, etc.

Indicates a repetition of a parenthesized list or the *nearest* conversion code letter (or a line-up code, but *not* an S or a string) to its left and is either an unsigned integer constant or a letter R denoting a 'remote' specification of value. The consecutive R's in the format string are associated left to right with the consecutive elements of an integer array or a list of procedure arguments (cf. Knuth (1964) sect. B.3.1), which can be preset before the activation of input/output and then counted down by one on each execution of the relevant item, the editing action ceasing when the count reaches zero. The method used is irrelevant here, except that if negative initial values were allowed, we would obtain the equivalent of an infinite loop terminated only when the input/output list has been exhausted. If so, then the action on a next call of the same format would be as described later (see: 'Format').

Subfield Separator: C2→S3 | {ϕ|S3} C {S3|ϕ}

Examples: SSS
C
'KG'S'M'SCS, etc.

In a numeric item this is either an insertion as discussed above or a single code C possibly surrounded by insertions. C will cause a comma to appear in the indicated position unless the nearest digit field to the left had a suppressed zero whence a blank or a fill-up character would be substituted for the comma.

Subfield: C5→C2 | {ϕ|C2} R1 {C2 R1}* {C2|ϕ}

Examples: 2
SCS
SC2SS2, etc.

This is either a subfield separator or an alternating sequence (in particular a single element) of replicators and subfield separators. All replicators refer to the *same* nearest left code letter or line-up mark and each one is individually interpreted as explained before.

Decimal Integer: D3→D {{ϕ|C5} D}* {C5|ϕ}

Examples: DD
D3SD3
D2C2C2, etc.

Decimal integer format starts with a D (to indicate decimal conversion) and continues with more D's or subfields separated by D's. Each D *not followed* immediately by a replicator stands for a single digit position in the edited number, while a D *followed* by replicator stands for as many positions as indicated by the value of that replicator. Thus D3S ≡ DDDS while DS3 ≡ DSDDD. This interpretation accords better with the intuitive meaning of code. Standard editing suppresses all leading (i.e. non-significant) zeros and places the sign of the number ('—' if negative or a blank if positive) in a D field immediately to the left of the first non-suppressed digit of the aligned number expansion; hence the integer format field width should be at least 2. If a value of zero is output either nothing will appear at all or the fill-up characters only will be printed, in case the item is within the scope of a prefix F, as explained later. A **real** number will be rounded-off to the nearest integer before editing. If the field width allowed is insufficient to accommodate the expansion an overflow condition will be raised; the action in such case is undefined.

Decimal Z-integer: D6→D3 Z {{ϕ|C5} D}* {C5|ϕ}

Examples: DDZ
DS3SZ3
DZC3C3S3, etc.

The purpose of decimal Z-integer format is to print the number including some or all of its leading zeros, i.e. this guarantees a visible entry regardless of the edited value. Z code acts as a trigger indicating that the leading zeros, normally suppressed, are to be printed starting with the digit position associated with the Z itself. Note that otherwise Z is treated within the format as a D, i.e. Z3 expands to ZDD. For further illustration see also the conversion examples at the end of this section.

Decimal Real: D9→{D3|D6} . {{ϕ|C5} D}* {C5|ϕ}

Examples: DD.3S2
D3S3.2S'KG'
D.DDDD, etc.

This consists of a leading decimal integer or decimal Z-integer subfield followed by a decimal point and possibly a fractional subfield specification. The latter has exactly the same pattern allowed for the integer part. If the output value is zero, the integer part will be suppressed and only the strings, which are parts of insertions, if any, will come out—unless zeros are forced by the Z trigger, or other fill-up characters by the prefix F. Fraction will be printed in any case.

Decimal Exponential:

D12→{D3|D6|D9} $_{10}$ {{ϕ|C5} D}* {C5|ϕ}

Examples: $D5_{10}3$
$D3S3.2_{10}2$, etc.

This format must start with either an integer, Z-integer or a real decimal field followed by a $_{10}$ and an exponent subfield constructed again according to the same pattern used in the earlier parts. For this type of editing there is never any zero suppression; the mantissa of number is aligned from the left, sign is placed in the first position and the exponent adjusted suitably. The prefix F code and the Z trigger will thus have no effect on editing.

Constructions similar to D3, D6, D9 and D12 can be defined for octal and binary conversions indicated by codes Q and B respectively. These are not discussed further but some examples will be found at the end of this section.

Alpha Format:

A4→A {{ϕ|S3} {R1 S3}* {ϕ|R1} A}*
cont. {ϕ|S3} {R1 S3}* {R1|ϕ}

Examples: A27
AAAS'INITIALS'S3SS'NAME'SS20, etc.

Alpha formats are used for the transmission of strings. Again the interpretation of replicators and insertions is as for the decimal number formats, except that C codes are not allowed here. The output of **Boolean** values would normally be done through the alpha formats, the conversions being specified by string procedures within the output list. If the transmitted string is shorter than the field width allowed, the edited string is padded on the right with blanks; if longer, then an overflow condition must be raised and the action is undefined. An F prefix over the alpha field has no meaning unless that is specified outside the standard system.

Editing Format:

E→D3|D6|D9|D12|Q3| . . . , etc. . . .|A4

This is simply a collective category for all format items specifying the editing of values as opposed to the 'line-up' formats not associated with editing.

Line-up Format:

L→{X|/|P} {{ϕ|R1} {STRN R1}*
cont. {STRN|ϕ} {X|/|P}}*
cont. {ϕ|R1} {STRN R1}* {STRN|ϕ}

Examples: X27
////
PX115'NOTES'/3, etc.

These formats specify the device form control and possibly the printing of constant titles. X code is used to indicate the skipping of spaces within a given line, / stands for a carriage return followed by a line skip, P will position the form at the top of the next physical page. Again the sequences of replicators separated by strings and trailing each code character are interpreted similarly to those trailing the editing codes. Note that S's and C's are not allowed as separators within the line-up replicator sequence as they would make no sense in this context. In place of S we can use X with the same effect; also X can be replicated while S can not.

† The examples for F1 and F5 will not be carried down to the level of basic symbols so as not to obscure the patterns.

Primary:

F1→{ϕ|PX} E | {ϕ|PX} ({ϕ|STRN} F5) R1
cont. {S3 R1}* {S3|ϕ}

Examples: †:E
PX E
(L E , L (E L E) R1 S3) R1, etc.

A primary is either an editing format possibly prefixed (PX) or a 'chain' (F5) of format items enclosed in parentheses, possibly preceded by a prefix and *always* followed by at least one replicator or an alternating sequence of replicators and insertions. Since each item in a chain may again be a primary, the definition is necessarily recursive. The prefix PX consists of a letter F (for: 'fill') followed by a single character code ('CHAR' in the syntax part) which indicates a particular editing function to be applied to all items within the scope of prefix, i.e. either a single editing format or the parenthesized list. In particular F* will fill all suppressed zeros and S's included within their sequence (but not the strings!) with asterisks. Other characters may specify special editing functions—these are undefined within the general system. The replicator (sequence) applies to the entire parenthesized expression and means a sequence of repetitions of fields generated by this expression, separated by blanks or insertions copied from the replicator sequence. This simplifies the specification of grouping on the edited line, e.g. (D2S)2SS2 expands to DDS,DDSXXDDS,DDS. Note that the automatic insertion of separating commas and the transformation of the replicator S's into X's is necessary since it is the parenthesized editing format that is to be repeated and not its component subfields.

Chain: F5→L | {ϕ|L} F1 {{L|,} F1}* {L|ϕ}

Examples: L
F1
L F1
L F1, F1 L F1, etc.

This is either a line-up format or an alternating sequence (possibly a single element) of format primaries, and either the line-up formats or commas, the latter used when there is no line-up action between the successive primaries. When primaries are separated by at least one blank, or a line or page skip, and when there are no insertion literals, the use of expanded line-up formats permits an exact match between the format and the output string. (See the last conversion example.)

Format: F9→' {ϕ|STRN} F5 '

Examples: 'P'CONT'//((D10.2X8)6/)56'
'//D3S3.2,(D2S2.2SS)2SS2', etc.

Finally, format is a chain of items possibly preceded by a literal and enclosed in string quotes. During execution, successive editing format items from left to right, allowing for parenthesized list repetitions, are used to edit the successive items of an input/output list while the line-up formats control the layout. In most situations the number of editing items in a format will probably be the same as the number of items to be edited in the input/output list; if not so then we have to consider three cases:

(*a*) The end of format is reached before the input/output list is exhausted—then the format should simply repeat from the beginning.

(*b*) The values list is exhausted before the end of format is reached—then the format should be executed *as far as possible* (i.e., including any line-up actions following the last editing item activated†), and on its next invocation, presumably with some other values list, it should start at the beginning again, and not at the point where it left off earlier.

(*c*) There are no values to be edited at all—then the format should execute only the line-up actions, if any, up to the first editing item.

Additional line-up actions may have to take place automatically where forced by the physical properties of the external medium (e.g. line or page overflow, tape end, etc.), no automatic line skip is invoked though merely because of repeated activation of the same format, hence a print line may be built up, piecemeal, by appending the edited text generated through those activations. The exact specification of the automatic actions caused by interrupts remains outside the scope of the article (cf. Knuth, 1964).

On input, the action of all defined above constructs is in general the same as during output, but less format conformance is required. Thus, for a numeric input a number located anywhere in the field indicated by the format would be accepted with its decimal point and/or $_{10}$ overriding the format specified positions, while the fields corresponding to literals in the format would simply be skipped; also a leading plus sign would be accepted in lieu of a blank.

A few general remarks should be made on the system as a whole: the syntax has been set up so as to allow a literal almost anywhere within the format string. The entire system is conceptually based on a kind of prefix coding with each character like D, A, X, /, P determining the meaning of all subsequent constructions trailing it, until a new interpretation is triggered by a different character. The last production (No. 166) is needed only for the parsing; symbol T being an arbitrary marker to help start and end the process simplifying thereby the analysing algorithm. At the end of parse the stack configuration is: T F9 T. The syntax still allows some rather unusual constructions which, while not wrong in that they permit unambiguous interpretation, would seldom if ever be used in practice. Thus a comma may precede or follow a period (D3C.C2 yields, say, 27,.,66 [*sic.*] Commas can also be specified within exponent fields, though it was unlikely that exponents huge enough to warrant it would ever be used. For comparison the report Knuth (1964) also allows some such constructions so that the syntax defined here is compatible with some of his proposals; on the other hand one could probably eliminate such cases by further elaboration of the production system.

† This takes care of a frequently encountered case when a line-up action is required *after* editing and is in line with sect.A.2.3.6. of Knuth (1964), but contrary to the current interpretation of formats in PL/1.

The examples showing the editing of numbers and demonstrating the flexibility of the proposed system have been collected in **Table 2**, while those given in **Table 3** illustrate the illegal constructions.

Table 2

Examples of format editing of numbers

In each block: 1st line is a 'short' format,
2nd line is an equivalent 'expanded' format
3rd line is the result from a value 80247.5625 dec = 234567.44 oct.
4th line is the result from a value −17.765625 dec = −21.61 oct.

```
D6
DDDDDD
 80248
   −18

D4.3
DDDD.DDD
Overflow
−17.766

D5₁₀3
DDDDD₁₀DDD
 8025₁₀+01
−1777₁₀−02

D.6₁₀3
D.DDDDDD₁₀DDD
 .802476₁₀+05
−.177656₁₀+02

D4S'KG'S3.1S'G'
    KG      G
DDDDS  SDDD.DS
  80 KG 247.6 G
     KG −17.8 G
```

```
D3C3.5
DDDCDDD.DDDDD
 80,247.56250
    −17.76562

 D2DSD3.2S2SD2
DDDSDDD.DDSDDSDD
 80 247.56 25 00
    −17.76 56 25

DZDC3C3.2
DZDCDDDCDDD.DD
 00,080,247.56
−00,000,017.77

Q4SSQQQ.2
QQQQSSQQQ.QQ
 234  567.44
      −21.61

DDDCDDD.DDXXXXDDD.DDDSDDD
 80,247.56    −17.765 625
```

```
B10.4
BBBBBBBBBB.BBBB
Overflow
    −10001.1100

B3.8₁₀5
BBB.BBBBBBBB₁₀BBBBB
 10.01110011₁₀+1111
−10.00111000₁₀+0011

D3SZ3S3.2
DDDSZDDSDDD.DD
    080 247.56
   −000 017.77

F*D3C3C3.2
F*DDDCDDDCDDD.DD
  *****80,247.56
  ********−17.77
  ::: Line picture
  ::: Edited line
```

Conclusions

It remains to compare the system as described above with the one defined in Knuth (1964) and serving as a yardstick, and in particular to discuss those features of the latter which are not available here:

A. Sign Control: The sign at the left of a number will, in general, be floating. The option to provide other sign conventions is deliberately left open through the possible specification of prefix code (see 'primary'). The sign *following* a number can be provided easily by means of alpha field and a corresponding output list item keyed to the signum function, and by outputting the absolute value of the number. Note, however, that the position on the left where the sign normally would have been, is wasted.
B. Truncation: The standard action here is to round the numbers. Truncation can be easily achieved by modifying the value of the list item itself, hence a separate control through the format is not really needed.
C. Virtual Decimal Point: This again can better be done through the multiplication of the value in the output list by a proper power of 10 so as to shift it left, rather than by complicating the format system. On input we might use the FORTRAN type convention: i.e. a decimal point, if present in the input stream, overrides the position specified by format; if not present, then a virtual point is assumed just ahead of the first digit field of format indicated fraction.
D. String Formats: These were evidently forced into Knuth's Report by the need to distinguish string literals appearing in the program from those input to the program. The need for any distinction should disappear, however, when string-handling facilities are added to ALGOL.
E. Boolean Formats and Non-formats: It did not seem worth while to introduce special codes for these. Boolean quantities can be mapped on input and output into the representing character strings, which are handled through alpha formats, while the non-formats for transmission of values in the internal machine form might perhaps be better considered as I/O commands not associated with any formats.
F. Indefinite Replication: No special construction covers this purpose although the effect could be obtained by either a sufficiently large remote element R, or by allowing the initial value of such element to be negative and then counting it down. The latter solution introduces some incompatibility, since unlike R, an explicit (literal) replicator cannot be negative. In any case, on exit from the loop through an end of file or termination of output list the next activation of the same format starts again at the beginning, as explained before.
G. Standard Formats: The choice to provide these is entirely arbitrary and does not affect the specification.

On balance, it would seem that sensible choices have been made as most features omitted can be compensated for by the modification of output values through expressions, while the facility to obtain at will either the octal or the binary editing, in addition to the decimal, might prove useful in practice. Coding of report generators should be easier with the possibility of exact one-to-one match of position with the expanded line picture, while the use of replicator sequences within editing items or after the parenthesized lists adds much flexibility and power to the specification. An 'escape' mechanism obtained through the use of prefix code might enable one to tailor relatively easily the general system to the particular needs, and the general idea of having the complexity of specification to reflect the complexity of desired editing, should provide all the necessary gradations of coding for the majority of situations encountered in practice.

Table 3

Examples of formats disallowed by the syntax

D3'P''Q'3	Adjacent strings not allowed.
F(D3.2)2	Unbalanced parentheses: The opening parenthesis (after the F) is parsed into the prefix.
(D3.3,D4.3),	At least one replicator must follow each parenthesized list.
D3R2SS.D	Replicators must be separated by insertions or D's.
D3XD3	This implies editing formats for *two* items. If only one item was intended, use D3SD3.
D3CSSC3	Comma insertion codes must be separated by replicators or D's.
.D3C3	There must be at least one numeric field ahead of decimal point to allow for the sign.
//SS'PAGE 1'S//	No S codes are allowed within the line-up formats, use: //X2'PAGE 1'X//.
X3, X5	The line-up formats may not be separated by commas.
DZ3Z3.2Z2	Only one Z may appear to the left of decimal point and none to the right.

References

BRZOZOWSKI, J. A. (1964). Derivatives of regular expressions, *JACM*, Vol. 11, p. 481.
IFIP/WG 2.1. (1964). Report on input-output procedures for ALGOL 60, *Comm. ACM*, Vol. 7, p. 628.
KNUTH, D. (Ed.) (1964). A proposal for input-output conventions in ALGOL 60, *Comm. ACM*, Vol. 7, p. 273.
WIRTH, N., and WEBER, H. (1966). A generalization of ALGOL, and its formal definition: Part I, *Comm. ACM*, Vol. 9, p. 13.
(1963). Revised report on the algorithmic language ALGOL 60, *The Computer Journal*, Vol. 5, pp. 349–367.
(1966). ASA FORTRAN, New York: American Standards Association.