APPENDIX 6. Syntax Charts

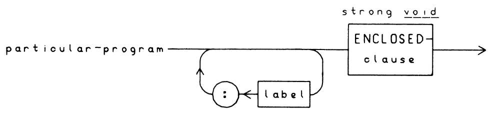
The following charts show exactly which sequences of symbols from a legal ALGOL 68 particular-program and which do not. To see what you may legally write, start where it says "particular-program" in the first chart below, and follow the line. Where the line diverges, you have a choice. You may either write an "ENCLOSED-clause", or you may write a "label" followed by a ":". If you write a label, then you get back where you started so, following the same lines again, you may now write an "ENCLOSED-clause" or you may go for another label. Eventually, you must write an ENCLOSED-clause in order to reach the outgoing arrow on the right, which signifies that your particular-program is complete.

In order to write any construct enclosed in a rectangle (such as an "ENCLOSED-clause"), you must find the start of that construct (usually on another chart) and follow the line from there, writing such constructs as you meet on the way, until you escape via an outgoing arrow. Then you have completed the construct in question and may continue following lines in the original chart. If you encounter a circle (or an oval), simply write the symbol inside it. So, to write an ENCLOSED-clause, find the start on the ENCLOSED-clauses chart. Immediately you are faced with a choice. Suppose you follow the route marked "closed-clause". Now you must write either "**begin**" or "*(*", and after that a "serial-clause" (which is on yet another chart). When your serial-clause is complete, you write "**end**" or "*)*", whereupon you reach the outgoing arrow and your ENCLOSED-clause is complete. Although the chart does not show it (it would have been just too complicated), if you write "**begin**" (rather than "*(*") before the serial-clause, then you must write "**end**" (rather than "*)*") after it, and vice-versa.

Every construct written inside a rectangle will thus be found as an entry point somewhere in one of the charts. The only exceptions are some very simple ones such as "label", "defining-identifier", "field-selector", "mode-indication", "operator", "character" and "digit". The first three of these are the same as "applied-identifier" (on the units chart). For mode-indications and operators see 1.3.2 and 4.3.

Above some of the rectangles there appears an indication of the mode that the construct inside is expected to yield, and the strength of its context (5.1.0.2) or whether it may be balanced (5.2.0.1). The mode written underneath an outgoing arrow tells you the mode of the construct you have just written. "MOID" stands for any mode including **void**, and "MODE" for any

PARTICULAR-PROGRAM

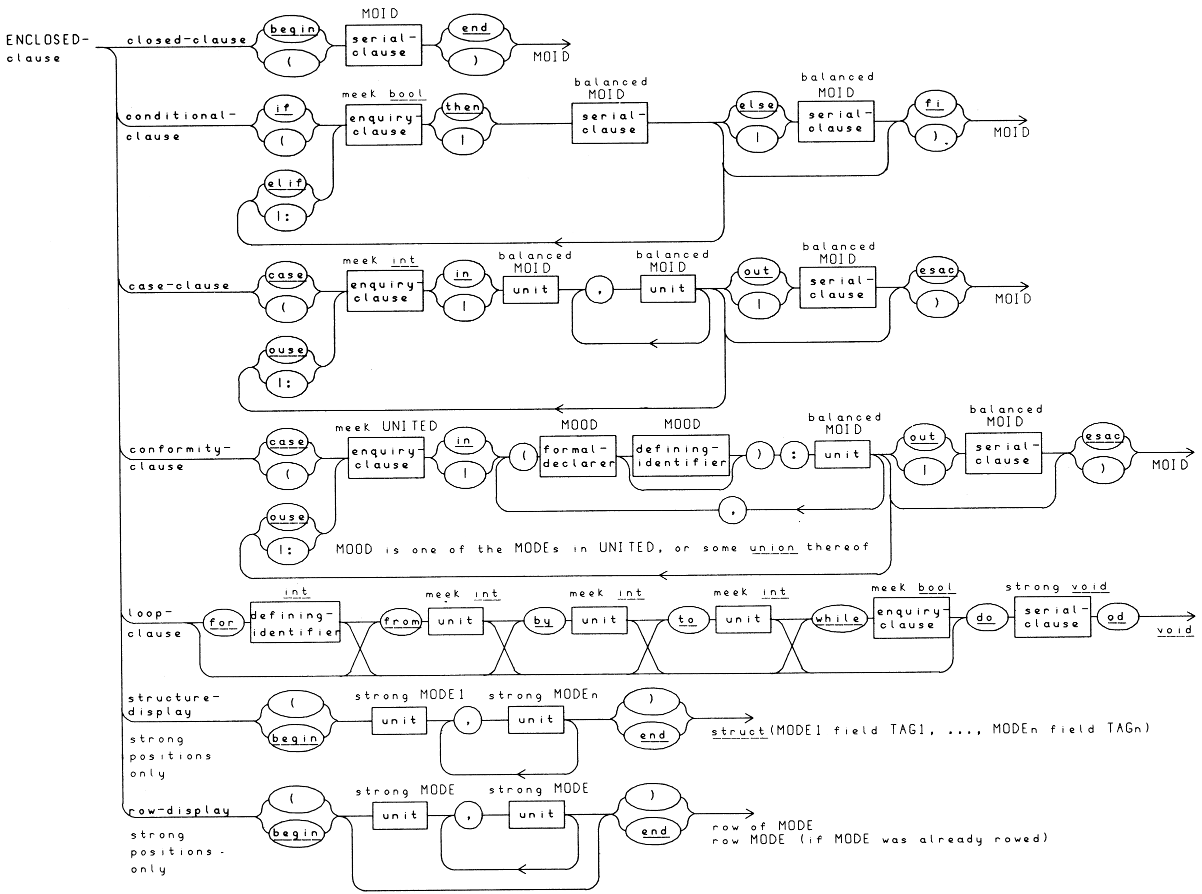


mode other than **void**. On any one pass through a particular chart, the MODEs etc. encountered must, however, always stand for the same mode.

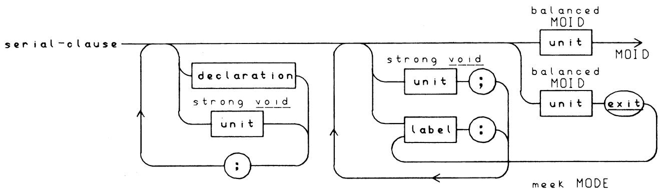
Generally speaking in ALGOL 68, comments and pragmats (1.3.2) may appear in between any two symbols, but there are some exceptions – notably in identifiers, denotations and format-texts. In these charts, you may insert a comment or a pragmat anywhere where you are following a continuous line, but if your route between two symbols is entirely over dotted lines, then you may not write comments or pragmats although blanks and newlines are still permitted (but see 5.5.1.1 for the dangers of doing this in string-denotations and see Appendix 5 for a commonly used solution to the problem).

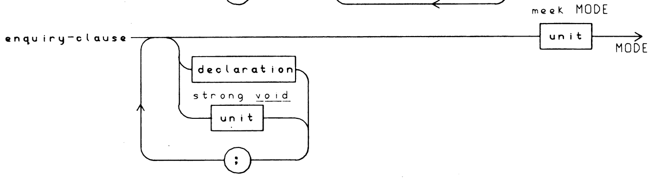
In "collection-lists" in the format-texts chart, an indication is given of the modes in the data list of *getf* and *putf* which are compatible with the various patterns. For example, the chart shows that for a real-pattern the mode in the data list on output may be **int** or **real**, but than on input it may only be **ref real**. See 7.6.1.3 for further details on this point.

ENCLOSED-CLAUSES

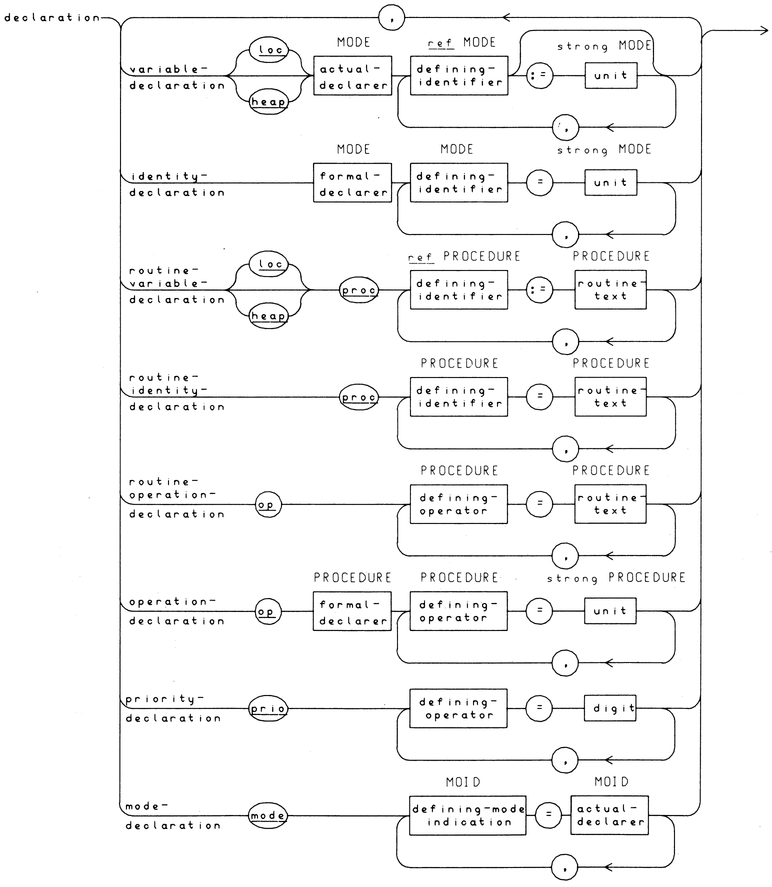


SERIAL-CLAUSES

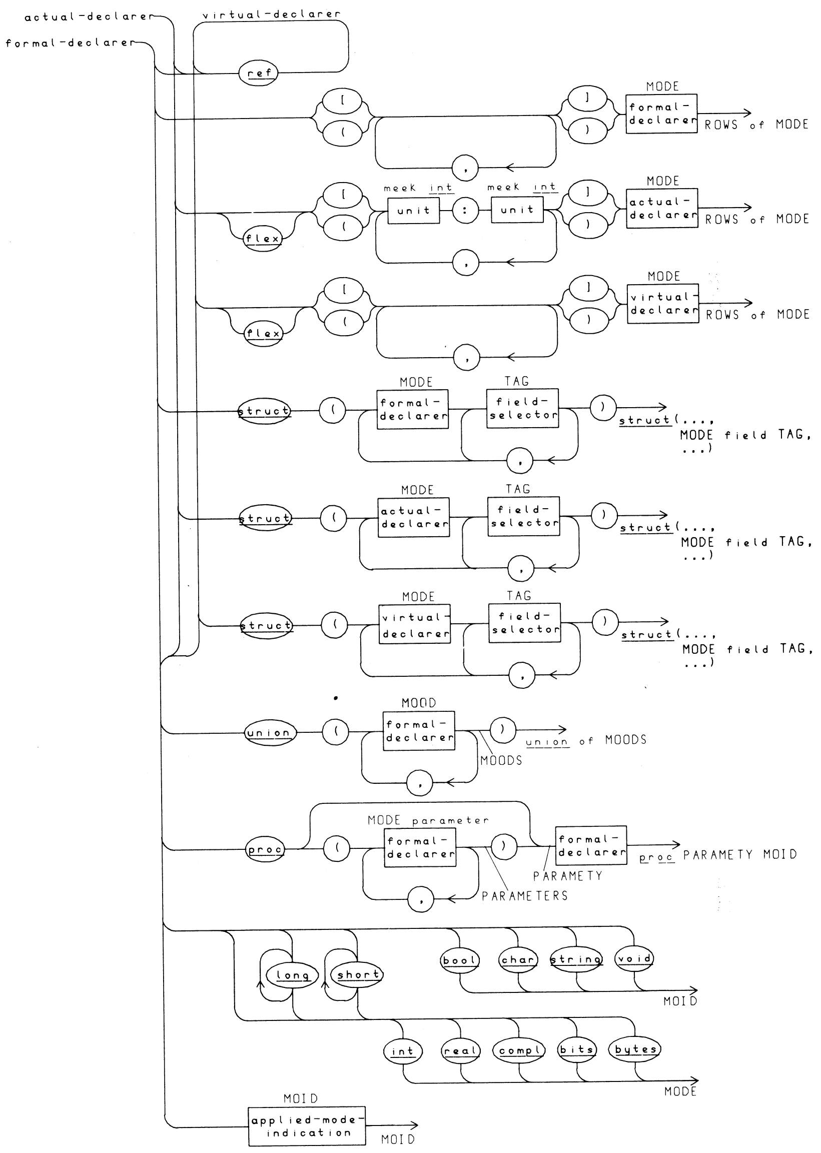




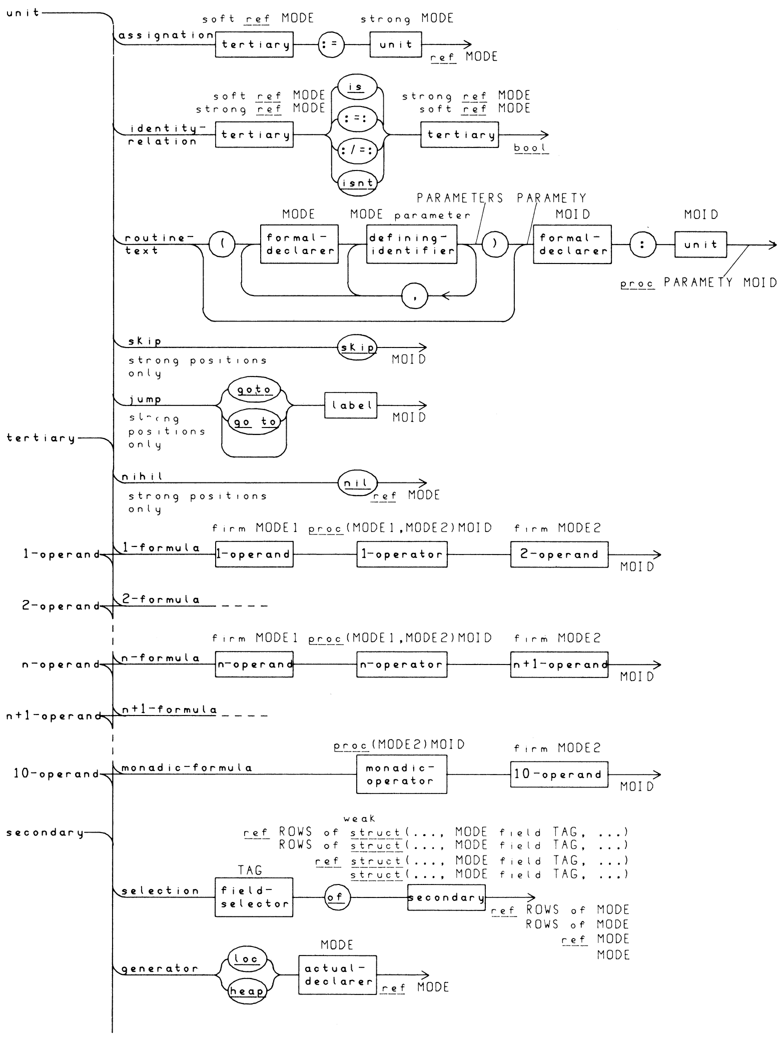
DECLARATIONS

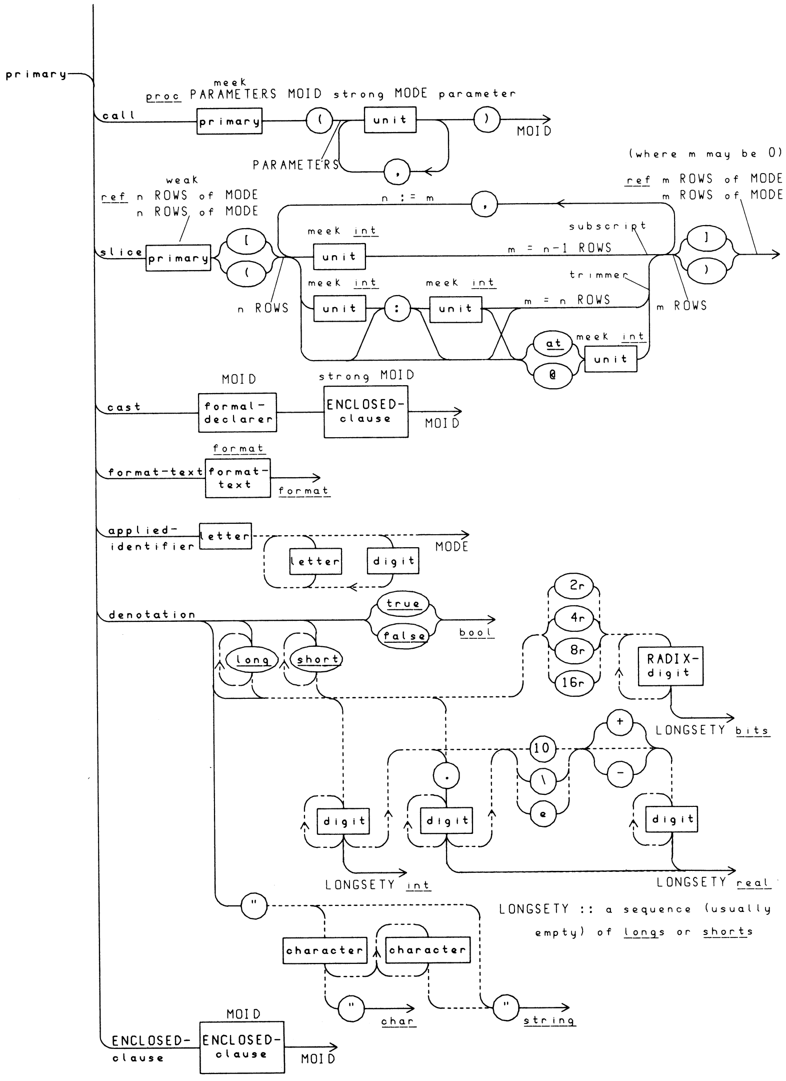


DECLARERS


actual-declarer
formal-declarer
virtual-declarer
ref
[
(
]
)
,
MODE
formal-declarer
ROWS of MODE
flex
meek int
unit
:
actual-declarer
virtual-declarer
struct
TAG
field-selector
struct (...., MODE field TAG, ...)
union
MOOD
MOODS
union of MOODS
proc
MODE parameter
PARAMETY
PARAMETERS
proc PARAMETY MOID
long
short
bool
char
string
void
MOID
int
real
compl
bits
bytes
MODE
applied-mode-indication
MOID

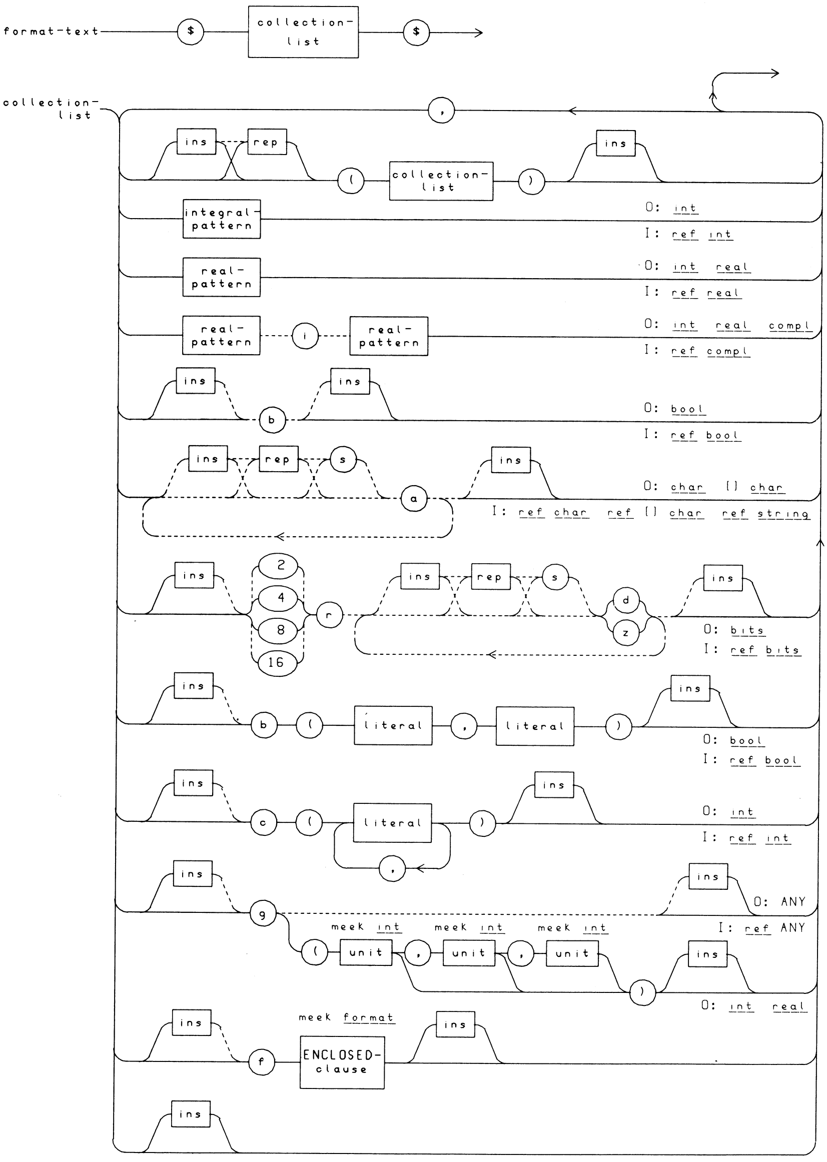
UNITS

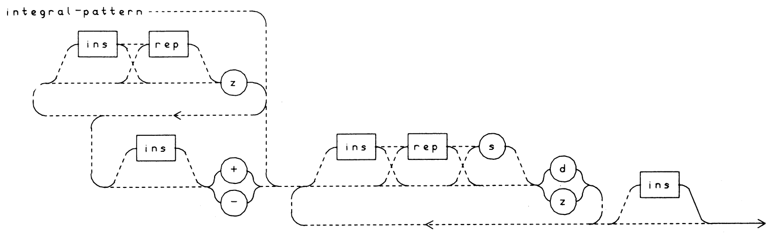

unit
assignation
soft ref MODE
tertiary
:=
strong MODE
unit
ref MODE
identity-relation
soft ref MODE
strong ref MODE
tertiary
is
:=:
:/=:
isnt
strong ref MODE
soft ref MODE
tertiary
bool
routine-text
(
MODE
formal-declarer
MODE parameter
defining-identifier
,
PARAMETERS
)
PARAMETY
MOID
formal-declarer
:
MOID
unit
proc PARAMETY MOID
skip
strong positions only
skip
MOID
jump
strong positions only
goto
go to
label
MOID
tertiary
nihil
strong positions only
nil
ref MODE
1-operand
1-formula
firm MODE1
1-operand
proc (MODE1,MODE2)MOID
1-operator
firm MODE2
2-operand
MOID
2-operand
2-formula
n-operand
n-formula
firm MODE1
n-operand
proc (MODE1,MODE2)MOID
n-operator
firm MODE2
n+1-operand
MOID
n+1-operand
n+1-formula
10-operand
monadic-formula
proc (MODE2)MOID
monadic-operator
firm MODE2
10-operand
MOID
secondary
selection
TAG
field-selector
of
weak
ref ROWS of struct(..., MODE field TAG, ...)
ROWS of struct(..., MODE field TAG, ...)
ref struct(..., MODE field TAG, ...)
struct(..., MODE field TAG, ...)
secondary
ref ROWS of MODE
ROWS of MODE
ref MODE
MODE
generator
loc
heap
MODE
actual-declarer
ref MODE

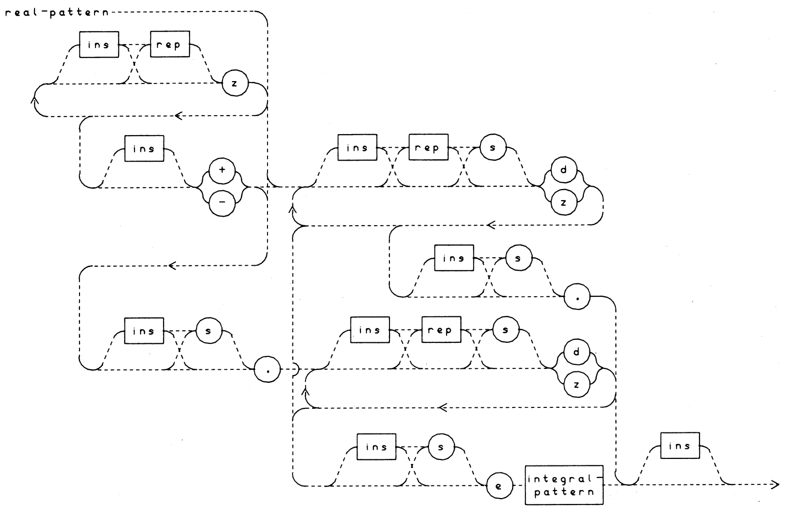


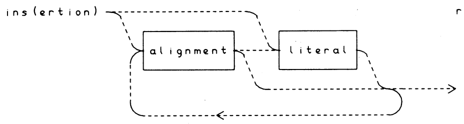
primary
call
meek
proc PARAMETERS MOID strong MODE parameter
primary
(
unit
)
,
PARAMETERS
MOID
(where m may be 0)
weak
ref n ROWS of MODE
n ROWS of MODE
ref m ROWS of MODE
m ROWS of MODE
n := m
subscript
slice
meek int
unit
m = n-1 ROWS
trimmer
[
]
n ROWS
m ROWS
:
m = n ROWS
at
@
cast
MOID
formal-declarer
strong MOID
ENCLOSED-clause
format-text
format
applied-identifier
letter
digit
MODE
denotation
true
false
bool
long
short
2r
4r
8r
16r
RADIX-digit
LONGSETY bits
10
\
e
+
-
.
LONGSETY int
LONGSETY real
"
character
char
string
LONGSETY :: a sequence (usually empty) of longs or shorts
ENCLOSED-clause

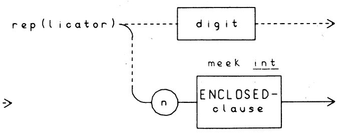
FORMAT-TEXTS

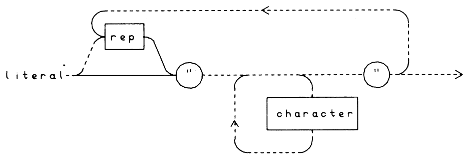


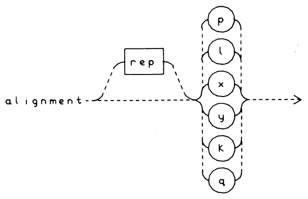

integral-pattern
ins
rep
z
ins
+
−
ins
rep
s
d
z
ins


real-pattern
ins
rep
z
ins
+
−
ins
rep
s
d
z
ins
s
.
ins
s
.
ins
rep
s
d
z
ins
s
e
integral-pattern
ins


ins(ertion)
alignment
literal


rep(licator)
digit
meek int
n
ENCLOSED-clause


literal
rep
"
character
"


alignment
rep
p
l
x
y
k
q