A Proposed Sublanguage of ALGOL 68

P.G. Hibbard
(Carnegie-Mellon University)

(Editor's Note -
Dr. Hibbard's Sublanguage Proposal has been discussed at several meetings of the Working Group and its Sub-committee on Sublanguages. The earliest form of the proposal was described in AB 35.3.2 and a later version was presented to the Dresden meeting of the Working Group (working paper (Dresden 3) 237). It was discussed again at the Cambridge meeting of the Sub-committee on ALGOL 68 Support (see the report of that meeting elsewhere in this issue). It has been implemented by Dr. Hibbard on a Modular One computer at the University of Liverpool, and is there in regular use for undergraduate teaching.

The Sublanguage is described in two documents. The first is a formal definition in the form of a list of changes to the Report. A first draft of this forms the bulk of the paper - when finalized it will be submitted to the Working Group for approval. The second will be a companion volume to the first and will describe the complete sublanguage in the style of the main Report, but with considerably less formality. This is at present in preparation, but the first few pages of it are included here in order to establish the aims of the proposal and to list the principal restrictions imposed).

Informal Description

0. Introduction

In designing this sublanguage of ALGOL 68, an effort has been made to produce a programming language which will be of value to many people in the computing community. Whilst it is intended primarily for programming in numerical and semi-numerical areas, it contains however many features which will be useful for other applications. The language is intended as a supplement to the full language ALGOL 68, not as a replacement for it, and it is hoped that its use will encourage the use of the full language.

The following considerations have led to the development of this language:

(a) If some features are removed from ALGOL 68 altogether, and the generality of other features restricted, the size and complexity of the compiler and the run-time system required for the language are considerably reduced. As a result the language may then be implemented on computers too small for feasible implementations of the full language, and it may also be used for specialised purposes on larger computers, e.g. for fast in-core compilation and execution of short programs.

(b) Many programmers do not require all the features of ALGOL 68. By incorporating into a sublanguage those features which are likely to be of use for certain applications, the task of learning and using the language is simplified. It is important, however, that the sublanguage should look, to the uninitiated user, like a complete programming language in its own right, and that programs in the intended application area should be closely similar to those which would be written in the full language by experienced programmers.

(c) Since it contains only those features which are likely to be present in many implementation subsets of ALGOL 68, the sublanguage can act as a reference standard for algorithms in certain fields, and thus lead to wider use of the language.
0.1. Comparison with ALGOL 60

This sublanguage has greater generality and power than ALGOL 60. Since it has been designed as a completely new language, and not as an expansion of ALGOL 60, it is difficult to isolate the differences between the two. However the following list should give insight into the more significant differences:

(a) Whereas ALGOL 60 has the primitive types integer, real and boolean, the sublanguage has in addition the "modes" char (character), string, compl (complex), bits and bytes, and different sizes of ints (integers), reals, compls, bits and bytes.

(b) In the sublanguage references (pointers) may be declared and manipulated. References provide an efficient and powerful parameter passing mechanism, and allow chains of indirect addresses to be built up.

(c) The sublanguage allows values to be collected up not only into multiples (arrays), but also into structures, in which the elements need not be of the same mode. Access to the individual elements in structures is possible using an efficient "field selection" mechanism. Structures and multiples may be assigned, passed as parameters, returned as values from procedures, and be themselves collected into more complicated structures and multiples, in just the same way as may the primitive values.

(d) The concept of subscriptions of arrays in ALGOL 60 is extended to allow "slicing" of subarrays from larger arrays, for example columns and rows from matrices. By appropriate use of references and declarations, highly efficient algorithms for matrix manipulations can be constructed.

(e) In the sublanguage it is possible to declare "mode indications" (boldface, or underlined words) which may then be used, for example, to declare variables. This improves the clarity and intrinsic understandability of sublanguage programs, and provides, through the use of references and structures, a means of performing list processing without the need for additional library facilities.

(f) Operation declarations in the sublanguage allow the introduction of new operators, and allow the class of operands of already existing operators to be extended.

(g) A number of procedures for input and output of single values, collections of values, multiples and structures is provided.

(h) The ALGOL 60 loop statement is replaced by a more efficient and concise repetitive statement.

(i) A number of language features of ALGOL 60, such as conditionals, blocks and procedure declarations, have been made more consistent in the sublanguage. Some other features, such as label parameters and switches, do not appear, but their expressive power is included in other, more general, constructions.

0.2 Comparison with ALGOL 68

Every program in the sublanguage is a program in the full language, with the same meaning. A program in the sublanguage is neither more defined nor less defined than the same program in the full language.

There is a sufficient number of features not included in the sublanguage for the advantages of expressing many algorithms in the full language to be obvious. The formal definition of this sublanguage gives a specification of these differences; the list below is a summary of them.
(a) Parallel-clauses may not appear in the sublanguage. Collateral-clauses may not appear in void contexts, or in certain balances.

(b) Unions and conformity-clauses do not occur in the sublanguage; however, certain operators and procedures of the standard prelude, which have parameters which are united modes, may appear.

(c) The defining occurrence if an indicator, unless it is a label-identifier, must precede the first applied occurrence of the indicator.

(d) Strings are treated as primitive values, and may not be sliced. Instead, procedures and operators are provided to manipulate substrings.

(e) The fields of a structure, and the elements of a multiple, may not themselves be multiples, though references to them are allowed. (An exception is made for strings).

(f) Only local-generators and non-flexible multiples are allowed. The heap-generators in the transput routines are handled using the same run-time facilities as are used for handling strings.

(g) A restricted number of transput declarations is provided. Formats and formatted transput may not be used.

Formal Definition

0. Introduction

This document is an addendum to the Revised Report on the Algorithmic Language ALGOL 68. It contains modifications and additions to the Report, which allow the Report then to define particular-programs both in the full language, without change, and in a proper subset of the full language.

Section 1 of this document is a commentary to assist the reader. Sections 2 and 3 specify, respectively, the changes which are required in the semantics of the Revised Report and the changes which are required in the preludes. Section 4 lists the declarations in the preludes of the Report which are to be "visible" (rule 7.2.1.ck) to particular-programs in the sublanguage, and Section 5 gives the changes and additions to the syntax.

No changes to the pragmatic remarks of the Report have been included, hence they will not be fully appropriate to the language defined here.

1. Commentary

1.1 The nest

The production rules of Section 5 place additional information in the nest of a construct. This information is as follows:

a) The flag "sublanguage" (1.2.3.BD). This occurs in the nest of a particular-program if it is descended from the second alternative of rule 10.1.1.g. The predicate 'where NEST indicates sublanguage' (2.2.1.aa,ab), which examines the nest for the flag, is used to select alternatives of those notions whose descendents in the sublanguage are different from those in the full language. Since the nest of the preludes does not contain the flag however, it is possible for the external environment of sublanguage particular-programs to be the same as that for full language particular-programs.
b) The counters "down TALLY" and "up TALLY" (1.2.3.AA,AB). These subdivide the program-text into sections which are consecutively numbered by the 'TALLY'. Each section ends with a defining-indicator. The 'TALLY' of a section is incorporated into the 'PROP' which corresponds to the indicator (4.8.1.a), and is used to ensure that the restrictions imposed on the sublanguage are satisfied (5.2.3.1.ab, 7.1.1.ai, 7.2.1.cb).

c) "protected TAB"s and "unprotected TAB"s (1.2.3.BD). The flag 'unprotected TAB' occurs in the nest of the actual-declarer of a definition of a mode-indication (4.2.1.b), and is changed into 'protected TAB' in the nest of the actual-bounds of such an actual-declarer (4.6.1.ha). It is used to prevent an applied-mode-indication occurring in the bounds of its actual-declarer (4.8.1.ba).

d) Layers andASHINGTONs (1.2.3.B,BB). A 'LAYER' is added to the nest for each construct which, when elaborated, will cause an environ to be established (Report 3.2.2.b). The 'ENVIRON' of a 'LAYER' is either 'nonlocal', when the corresponding environ is nonlocal, or it is 'local' or 'found', when it is local. The 'ENVIRON' is determined by the syntax (3.2.1.a, 3.4.1.j,k, 3.5.1.a,b, 4.2.1.b, 5.2.3.1.a,b, 5.4.1.1.b, 5.4.2.1.a, 5.4.3.1.a), and is used to restrict the occurrences of local-generators in sublanguage particular-programs (5.2.3.1.aa).

1.2. The syntactic positions

Certain constructs, e.g. collateral-clauses, may not occur in all the positions in which they may in the full language. Restrictions have been imposed by introducing a further 'SORT', i.e. 'unconditionally strong' (1.2.2.C,CA), and by modifying the rules for balancing (3.2.1.f). Within the full language no distinction is made between 'strong' and 'unconditionally strong' positions, but blind alleys may be produced in the sublanguage unless a position is 'unconditionally strong' (3.3.1.a,c,d,e, 5.2.3.1.aa).

1.3. Independence of properties

Some of the tests for independence in sublanguage particular-programs require that all the 'PROPS' in the nest are examined (and not simply those 'PROPS' in some 'LAYER'). They are collected together by production rules 7.1.1.af,ag. The tests are different, depending upon the defining-indicator.

a) A 'DYADIC TAD' (7.1.1.ab) is required to be independent of all 'PROPS' in the nest (7.1.1.bc,bd,c). This implies that the priority of a dyadic-operator may not be changed in an inner range. If the 'TAD' is not a 'TAB', then it is also required to have been defined as a dyadic-operator in an outer range, and hence, ultimately, in the preludes (7.1.1.ai,am,an,ao).

b) A 'DUO TAD' (7.1.1.ac) is required to be independent of all 'DUO TAD's in the collected 'PROPS' of the nest (see (f)), and to follow, textually, a priority declaration of the 'TAD' (7.1.1.ai,ai).

c) A 'MONO TAM' (7.1.1.ad) is required to be independent of all 'MONO TAM's in the collected 'PROPS' of the nest (see (f)), and to have been defined as a monadic-operator in an outer range, and hence, ultimately, in the preludes (7.1.1.ap,aq,ar,as).

d) A 'TALLY TAB' (7.1.1.ah) is tested for independence by examining the 'PROPS' of the 'LAYER'. The collected 'PROPS' of the nest are examined to ensure that the 'TAB' does not occur as a 'TAD' in an outer range (7.1.1.ai,ak).
e) A 'MABEL TAG' (7.1.1.aa) has the same tests for independence as in the full language.

f) If the sublanguage is indicated in the nest, the "firmly related" test for the independence of operators is replaced by a "meekly related" test (7.1.1.i,j). This requires that the modes of the corresponding parameters of any two independent operators cannot be deprefed to one same mode (7.1.1.na,nb).

1.4. Identification

a) The 'FLAG', if any, enveloped by the 'LAYER' in which an applied-indicator-with-TAX has been identified is extracted from the nest (7.2.1.a) to determine whether the applied-indicator is a protected mode-indication (4.8.1.ba).

b) The "TALLY" of the 'TALLY QUALITY TAX' which is identified in the nest by an applied-indicator, is compared with the 'TALLY' in the 'DOWN UP STACK' (7.1.1.c,cb), to ensure that the defining-indicator has preceded the applied-indicator.

c) If a QUALITY-NEST-applied-indicator-with-TAX in a sublanguage particular-program identifies a defining-indicator in the preludes (7.2.1.cc), additional tests are performed. These are as follows: 1) The 'QUALITY' is tested to ensure that it is "acceptable" (7.2.1.cd). It will be acceptable if it is 'label' or 'DYADIC', or if it is 'MOID TALLYX' and 'MOID' is either "declarable" in the sublanguage (7.2.1.ce,cf,cf,ch,cf,cf), or is a 'SPECIAL STRUCTURE' (7.2.1.AB), or is a procedure with 'SPECIAL PARAMETERS' (7.2.1.AC). 2) The 'QUALITY TAX' is tested to ensure that it is visible to the sublanguage (7.2.1.ck). The predicate 'where QUALITY TAX visible through window' permits a subset only of the defining-indicators of the preludes to be used in sublanguage particular-programs.

1.5. Strings

The mode 'STRING' (6.5.1.AA) is a structured mode with a flexible-row-of-character field, selected by a hidden field selector. If the nest indicates the sublanguage, a 'string denotation' (8.3.1.b) is of mode 'STRING' (8.3.1.aa), otherwise it is of mode 'row of character' (8.3.1.a). A value of mode 'STRING' cannot be sliced or deflexed to a row-of-character value. The particular-prelude (Report 10.5.1) is augmented to provide a mode declaration for the sublanguage, and to provide definitions of operators and procedures for manipulating 'STRING' values.

2. Modifications to the semantics of the Report

a) page 43 2.2.2.a+1
   "strong-void-new-closed-clause =>
       unconditionally-strong-void-down-i-UP-new-local-closed-clause #

b) page 48 3.2.2.b.+4
   # new => NEW FLAGETY #

c) page 48 3.2.2.+25:+27
   # If each 'PROP' "nonlocal" =>
      If the 'ENVIRON' enveloped by 'NEW' is 'nonlocal', then E is said to be "nonlocal" #
d) page 57 3.5.2.+12
    * -new- => -new-nonlocal-

e) page 57 3.5.2.+14
    * 'integral' => 'TALLY integral

f) page 57 3.5.2.+33
    * 'integral' => 'TALLY integral

g) page 67 4.8.2.+2
    * 'QUALITY' => 'TALLY QUALITY

h) page 67 4.8.2.+7
    * 'QUALITY' => 'TALLY QUALITY

i) page 67 4.8.2.+9
    * 'QUALITY' => 'TALLY QUALITY

j) page 87 6.5.2.+13
    * =>
    Case D: 'MODE' is 'STRING':
    W is the structured value whose field is built from V.

k) page 92 7.2.2.a.+2:+3
    * where-PROP-identified-in-NEST-LAYER =>
    where-NEST1-PROP-identified-in-NEST2-LAYER-with-FLAGETY

l) page 92 7.2.2.a.+3:+4
    * where-PROP-identified-in-NEST =>
    where-NEST1-PROP-identified-in-NEST2-with-FLAGETY

m) page 93 7.2.2.c+11
    * 'where QUALITY TAX resides in PROPSETY' =>
    'where NEST TALLY QUALITY TAX resides in PROPSETY

n) page 104 8.3.2.+1:+2
    * The yield ... as follows: =>
    a) The yield of a string-denotation D is determined as follows:
    Case A: The string-denotation is a row-of-character-denotation:
    The yield is the multiple value V "constituted from"{b} the string
    items contained in D.
    Case B: The string-denotation D is a STRING-denotation:
    The yield is a structured value whose field is a multiple value V
    constituted from{b} the string items contained in D.
    b) The value V constituted from the string items is determined as follows:

o) page 196 10.5.1.+6:+10
    * However ... contradicted) =>
3. Modifications to the preludes of the Report

a) page 154 10.3.2.2.b+2:+3
   # none of which contains => none of which, except 'STRING', contains #

b) page 197 10.5.1.i+1
   # =>

10.5.1.1. The sublanguage particular prelude

a) mode string = struct ( flex [ 1 : 0 ] char f1 );
b) op < upb , ] = ( string a ) int : 1 upb f1 of a ;
c) proc stringpack = ( [ ] char c ) string : ( string s ; f1 of s := c ; s );
d) op + = ( char c , b ) string : stringpack ( ( a , b ) );
e) op < , lt = ( string a , b ) bool : f1 of a < f1 of b ;
f) op <= , le = ( string a , b ) bool : not ( b < a );
g) op < , eq > = ( string a , b ) bool : a <= b and b <= a ;
h) op = , /= , de > = ( string a , b ) bool : not ( a = b );
i) op < , ge = ( string a , b ) bool : b <= a ;
j) op < , gt = ( string a , b ) bool : b < a ;
k) op R = ( string a , char b ) bool : a R string ( b );
l) op R = ( char a , string b ) bool : string ( a ) R b ;
m) op + = ( string a , b ) string : stringpack ( f1 of a + f1 of b );
n) op + = ( string a , char b ) string : a + string ( b );
o) op + = ( char a , string b ) string : string ( a ) + b ;
p) op < x , * >= ( string a , int b ) string : ( string c ; to b do c := c + a od ; c );
q) op < x , * >= ( int a , string b ) string : b * a ;
r) op < x , * >= ( char a , int b ) string : string ( a ) * b ;
s) op < x , * >= ( int a , char b ) string : b * a ;
t) op < ::= , plusab > = ( ref string a , string b ) ref string : a := a + b ;
u) op < ::= , plusoto > = ( string a , ref string b ) ref string : b := a + b ;
v) op < ::= , plusab > = ( ref string a , char b ) : a ::= string ( b );
w) op < ::= , plusoto > = ( char a , ref string b ) ref string : string ( a ) ::= b ;
x) op < x := , ::= , timesab > = ( ref string a , int b ) ref string : a ::= a * b ; #

4. The visible QUALITY TAXs

The QUALITY-TAXs for which the predicate 'where QUALITY TAX visible through window' holds are a subset of those defined by the declarations in the standard-prelude and particular-prelude of the Report, as augmented by this document. This subset is indicated below by listing the references of the appropriate declarations. This list may be extended to include more declarations from the preludes, at the discretion of the implementer (see production rules 7.2.1.AD,AE,AF,AG), provided that their 'QUALITYs' are acceptable (7.2.1.cd).

If a reference below is to a number of declarations, which have been produced by making the alterations specified in the Report 10.1.3, then all of the declarations so produced are to be regarded as being in the subset, with the following exceptions: a) If a number of different forms
1.2.3.

A) NEST :: DOWN UP STACK.
AA) DOWN :: down TALLY.
AB) UP :: up TALLY.
AC) STACK :: LAYER ; STACK LAYER.
B) LAYER :: NEW FLAGETY DECSETY LABSETY.
BA) NEW :: new ENVIRON.
BB) ENVIRON :: local ; nonlocal ; found.
BC) FLAGETY :: FLAG ; EMPTY.
BD) FLAG :: subconscious ; unprotected TAB ; protected TAB.
E) DEC :: TALLY1 MODE TAG[942A] ; TALLY1 priority Prio TAD[942f] ;
TALLY1 MOID TALLY2 TAB[942D] ; TALLY1 DUO TAD[942F] ; TALLY1 MONO TAM[942K].
K) LAB :: TALLY label TAG[942A].

1.3.1.

fa) where THING1 implies THING2 : where THING1, where THING2 ; unless THING1.
na) where <NOTETY1> is not <NOTETY2> : unless <NOTETY1> is <NOTETY2>[g].
nb) unless <NOTETY1> is not <NOTETY2> : where <NOTETY1> is <NOTETY2>[g].

2.2.1.

a) program :
unconditionally strong void down I UP new local closed clause[31a].
aa) WHETHER NEW FLAGETY PROPSETY indicates sublanguage :
WHETHER FLAGETY indicates sublanguage[ac,ad,-].
ab) WHETHER NEST NEW FLAGETY PROPSETY indicates sublanguage :
where FLAGETY indicates sublanguage[ac,ad,-], WHETHER true ;
unless FLAGETY indicates sublanguage[ac,ad,-],
WHETHER NEST indicates sublanguage[aa,ab].
ac) WHETHER DOWN UP indicates sublanguage : WHETHER false.
ad) WHETHER FLAGETY indicates sublanguage :
WHETHER <FLAGETY> is <sublanguage>. 

3.0.1.

c) * statement : unconditionally strong void NEST UNIT[5A].

3.1.1.

a) SOID NEST closed clause :
SOID NEST prefaced serial clause defining LAYER[32a] PACK.

3.2.1.

AA) PRACETY :: prefaced ; EMPTY.

a) SOID NEST PRACETY serial clause defining NEW FLAGETY PROPSETY :
where NEW determined by PROPSETY and NEST[aa],
SOID NEST NEW FLAGETY PROPSETY PRACETY series with PROPSETY(b).
aa) where new ENVIRON determined by DECSETY LABSETY and NEST:
    where <DECSETY> is <EMPTY>, where <ENVIRON> is <nonlocal>;
    where <DECSETY> is <DECS>, where NEST indicates sublanguage{22aa,ab},
    where <ENVIRON> is <local>;
    where <DECSETY> is <DECS>, unless NEST indicates sublanguage{22aa,ab},
    where ENVIRON determined by DECS{ab,ac}.
ab) where ENVIRON determined by EMPTY: where <ENVIRON> is <nonlocal>.
ac) where ENVIRON determined by DEC DECSETY:
    where <DEC> is <DYADIC TAD>, where ENVIRON determined by DECSETY{ab,ac};
    unless <DEC> is <DYADIC TAD>, where <ENVIRON> is <local>.

b) SOID DOWN UP STACK PRACETY series with PROPSETY:
    where <PRACETY> is <prefaced>, strong void DOWN up TALLY STACK unit{d},
    go on token, SOID down TALLY UP STACK series with PROPSETY{b};
    where <PRACETY> is <EMPTY>,
    unconditionally strong void DOWN up TALLY STACK unit{d},
    go on token, SOID down TALLY UP STACK series with PROPSETY{b};
    where <PROPSETY> is <DECS DECSETY LABSETY>,
    DOWN up TALLY STACK declaration of DECS{41a},
    go on token, SOID down TALLY UP STACK series with DECSETY LABSETY{b};
    where <PROPSETY> is <LAB LABSETY>,
    DOWN up TALLY STACK label definition of LAB{c},
    SOID down TALLY UP STACK series with LABSETY{b};
    where <PRACETY> is <EMPTY>,
    where <PROPSETY> is <LAB LABSETY> and SOID balances SOID1 and SOID2{e},
    SOID1 DOWN up TALLY1 NEST unit{d}, completion token,
    down TALLY1 up TALLY2 STACK definition of LAB{c},
    down TALLY2 UP STACK series with LABSETY{b};
    where <PROPSETY> is <EMPTY>, SOID DOWN UP STACK unit{d}.

f) WHETHER SORT balances SORT1 and SORT2:
    where <SORT> is <unconditionally strong>,
    WHETHER <SORT1> is <unconditionally strong> and <SORT2> is <unconditionally strong>;
    unless <SORT> is <unconditionally strong>,
    where <SORT1> is <strong>, WHETHER <SORT2> is <SORT>;
    unless <SORT> is <unconditionally strong>,
    where <SORT2> is <strong>, WHETHER <SORT1> is <SORT>.

i) * establishing clause:
    SOID NEST PRACETY serial clause defining LAYER{32a};
    MODE NEST PRACETY enquiry clause defining LAYER{34c}.

3.3.1.

a) UNCSTR void NEST collateral clause:
    unless NEST indicates sublanguage{22aa,ab}, UNCSTR void NEST joined portrait{b} PACK.
b) SOID DOWN UP STACK joined portrait:
    where SOID balances SOID1 and SOID2{32e}, SOID1 DOWN up TALLY STACK unit{32d},
    and also token,
    SOID2 down TALLY UP STACK unit{32d}
    or alternatively SOID2 down TALLY UP STACK joined portrait{b}.
c) UNCSTR void NEST parallel clause:
unless NEST indicates sublanguage{22aa,ab}, parallel token,
UNCSTR void NEST joined portrait{b} PACK.

d) UNCSTR ROWS of MODE NEST collateral clause:
where NEST indicates sublanguage{22aa,ab} implies <UNCSTR> is <unconditionally strong>,
where <ROWS> is <row>, UNCSTR MODE NEST joined portrait{b} PACK;
where NEST indicates sublanguage{22aa,ab} implies <UNCSTR> is <unconditionally strong>,
where <ROWS> is <row ROWS1>,
UNCSTR ROWS1 of MODE NEST joined portrait{b} PACK;
where NEST indicates sublanguage{22aa,ab} implies <UNCSTR> is <unconditionally strong>,
EMPTY PACK, where <NEST> is <down TALLY up TALLY STACK>.
e) UNCSTR structured with FIELDS FIELD mode NEST collateral clause:
where NEST indicates sublanguage{22aa,ab} implies <UNCSTR> is <unconditionally strong>,
NEST FIELDS FIELD portrait{f} PACK.
f) DOWN UP STACK FIELDS FIELD portrait:
DOWN up TALLY STACK FIELDS portrait{f,g},
and also token, down TALLY UP STACK FIELD portrait{g}.
g) NEST MODE field TAG portrait:
unconditionally strong MODE NEST unit{32d}.

h) * structure display:
UNCSTR structured with FIELDS FIELD mode NEST collateral clause{e}.
i) * row display:
UNCSTR ROWS of MODE NEST collateral clause{d}.
j) * display: UNCSTR STOWED NEST collateral clause{d,e}.

3.4.1.

b) SOID DOWN UP STACK1 chooser choice using MODE STYLE clause:
MODE DOWN up TALLY STACK1 prefaced enquiry clause defining LAYER2{c, -},
SOID down TALLY UP STACK1 LAYER2 alternate choice using MODE STYLE clause{d}.
c) MODE NEST1 PRACETY enquiry clause defining NEW DECSETY2:
where NEW determined by DECSETY2 and NEST1{32aa},
meek MODE NEST1 NEW DECSETY2 PRACETY series with DECSETY2{32b}.
d) SOID DOWN UP STACK2 alternate CHOICE STYLE clause:
SOID DOWN UP STACK2 in CHOICE STYLE clause{e};
where SOID balances SOID1 and SOID2{32e},
SOID1 DOWN up TALLY STACK2 in CHOICE STYLE clause{e},
SOID2 down TALLY UP STACK2 out CHOICE STYLE clause{i}.
h) SOID DOWN UP STACK2 in part of choice using UNITED:
SOID DOWN UP STACK2 case part of choice using UNITED{i};
where SOID balances SOID1 and SOID2{32e},
SOID1 DOWN up TALLY STACK2 case part of choice using UNITED{i},
and also token, SOID2 down TALLY UP STACK2 in part of choice using UNITED{h}.
i) SOID DOWN UP STACK2 case part of choice using UNITED:
MOID DOWN up TALLY STACK2 LAYER3 specification defining LAYER3{j,k, -},
where MOID unites to UNITED{64b}, SOID down TALLY UP STACK2 LAYER3 unit{32d}.
j) MODE NEST3 specification defining new nonlocal MODE TAG3:
NEST3 declarative defining new nonlocal MODE TAG3{541e} brief pack, colon token.
k) MOID NEST3 specification defining new nonlocal EMPTY:
formal MOID NEST3 declarer{46b} brief pack, colon token.
3.5.1.

a) unconditionally strong void DOWN UP STACK1 loop clause :
   DOWN up TALLY1 STACK1 for part defining new nonlocal integral TAG2{b},
   down TALLY1 up TALLY2 STACK1 intervals{c},
   down TALLY2 UP STACK1 repeating part with integral TAG2{e}.

b) NEST1 for part defining new nonlocal integral TAG2 :
   for token,
   integral NEST1 new nonlocal TALLY integral TAG2 defining identifier with TAG2{48a};
   where <TAG2> is <letter aleph>, EMPTY,
   where <NEST1> is <down TALLY up TALLY STACK1>.

c) DOWN UP STACK1 intervals :
   optional DOWN up TALLY1 STACK1 from part{ca},
   optional down TALLY1 up TALLY2 STACK1 by part{ca},
   optional down TALLY2 UP STACK1 to part{ca}.

d) optional NEST1 FROBYT :
   NEST1 FROBYT part{d};
   EMPTY, where <NEST1> is <down TALLY up TALLY STACK1>.

e) NEST1 repeating part with DEC2 :
   NEST1 new nonlocal DEC2 while do part{f};
   NEST1 new nonlocal DEC2 do part{h}.

f) DOWN UP STACK2 while do part :
   DOWN up TALLY STACK2 while part defining LAYER3{g},
   down TALLY UP STACK2 LAYER3 do part{h}.

h) NEST3 do part :
   do token, unconditionally strong void NEST3 serial clause defining LAYER4{32a}, od token.

4.1.1.

a) DOWN UP STACK declaration of DECS :
   DOWN UP STACK COMMON declaration of DECS{42a,43a,44a,e,45a,-} ;
   where <DECS> is <DECS1 DECS2> ,
   DOWN up TALLY COMMON declaration of DECS1{42a,43a,44a,e,45a,-},
   and also token, down TALLY UP STACK declaration of DECS2{a}.

b) DOWN UP STACK COMMON joined definition of PROPS PROP :
   DOWN up TALLY STACK COMMON joined definition of PROPS{b,c}, and also token,
   down TALLY UP STACK COMMON joined definition of PROP{c}.

4.2.1.

b) DOWN UP STACK mode definition of MOID TALLY TAB :
   where <TAB> is <bold TAG> or <STACK> is <NEW LAYER>,
   where DOWN UP STACK indicates sublanguage{22aa,ab} implies <MOID> is not <void>,
   MOID TALLY DOWN up TALLY1 STACK defining mode indication with TAB{48a},
   is defined as token,
   actual MOID TALLY down TALLY1 UP STACK new found unprotected TAB declarer{c}.

4.4.1.

a) DOWN UP STACK MODINE identity declaration of DECS :
formal MODINE DOWN up TALLY STACK declarer{b,46b},
down TALLY UP MODINE identity joined definition of DECS{41b,c}.
b) VICTAL routine down TALLY up TALLY STACK declarer:
procedure token.
c) DOWN UP STACK MODINE identity definition of MODE TAG:
MODE DOWN up TALLY STACK defining identifier with TAG{48a}, is defined as token,
MODE down TALLY UP STACK source for MODINE{d}.
e) DOWN UP STACK reference to MODINE variable declaration of DECS:
where DOWN UP STACK indicates sublanguage{22aa,ab} implies <LEAP> is <local>,
reference to MODINE DOWN up TALLY STACK LEAP sample generator{523b},
donw TALLY UP STACK reference to MODINE variable joined definition of DECS{41b,c}.
f) DOWN UP STACK reference to MODINE variable definition of reference to MODE TAG:
reference to MODE DOWN up TALLY STACK defining identifier with TAG{48a},
becomes token, MODE down TALLY UP STACK source for MODINE{d}.
where <MODINE> is <MODE>,
reference to MODE DOWN UP STACK defining identifier with TAG{48a}.

4.5.1.
a) DOWN UP STACK MODINE operation declaration of DECS:
operator token, formal MODINE DOWN up TALLY STACK plan{b,46p,-},
donw TALLY UP STACK MODINE operation joined definition of DECS{41b,c}.
b) formal routine down TALLY up TALLY STACK plan : EMPTY.
c) DOWN UP STACK MODINE operation definition of PRAM TAO:
PRAM DOWN up TALLY STACK defining operator with TAO{48a},
is defined as token, PRAM down TALLY UP STACK source for MODINE{44d}.

4.6.1.
BA) VIRMAL :: virtual ; formal.
e) VICTAL FIELDS DOWN UP STACK portrayer of FIELDS1:
VICTAL MODE DOWN up TALLY1 STACK declarer{a,b},
where DOWN UP STACK indicates sublanguage{22aa,ab}
implies <MODE> is not <ROWS of MODE1>,
donw TALLY1 UP STACK MODE FIELDS joined definition of FIELDS1{41b,c}.
where <FIELDS1> is <FIELDS2 FIELDS3>,
VICTAL MODE DOWN up TALLY1 STACK declarer{a,b},
where DOWN UP STACK indicates sublanguage{22aa,ab}
implies <MODE> is not <ROWS of MODE1>,
donw TALLY1 up TALLY2 STACK MODE FIELDS joined definition of FIELDS2{41b,c},
and also token, VICTAL FIELDS down TALLY2 UP STACK portrayer of FIELDS3{e}.
g) VIRACT flexible ROWS of MODE NEST declarator:
unless NEST indicates sublanguage{22aa,ab},
flexible token, VIRACT ROWS of MODE NEST declarer{a}.
h) VICTAL ROWS of MODE DOWN UP STACK declarator:
where DOWN UP STACK indicates sublanguage{22aa,ab}
implies <MODE> is not <ROWS1 of MODE1>,
protected VICTAL ROWS DOWN up TALLY STACK rower{ha,hb} STYLE bracket,
VICTAL MODE down TALLY UP STACK declarer{a,b}.
ha) protected actual ROWS NEST NEW FLAGETY rower:
   where <FLAGETY> is <unprotected TAB>,
   actual ROWS NEST NEW protected TAB rower\{i,j\};
   unless <FLAGETY> is <unprotected TAB>, actual ROWS NEST NEW FLAGETY rower\{i,j\}.

hb) protected VIRMAL ROWS NEST rower:
    VIRMAL ROWS NEST rower\{i,k,l\}.

i) VICTAL row ROWS DOWN UP STACK rower:
   VICTAL row DOWN up TALLY STACK rower\{i,j,k,l\}, and also token,
   VICTAL ROWS down TALLY UP STACK rower\{i,j,k,l\}.

j) actual row DOWN UP STACK rower:
   DOWN up TALLY STACK lower bound\{m\}, up to token,
   down TALLY UP STACK upper bound\{n\};
   DOWN UP STACK upper bound\{n\}.

k) virtual row down TALLY up TALLY STACK rower: up to token option.

l) formal row down TALLY up TALLY STACK rower: up to token option.

s) VICTAL union of MOODS1 MOOD1 mode NEST declarator:
   unless NEST indicates sublanguage\{22aa,ab\},
   unless EMPTY with MOODS1 MOOD1 incestuous\{47f\},
   union of token, MOIDS NEST joined declarer\{t,u\} brief pack,
   where MOIDS ravel to MOODS2\{47g\}
   and safe MOODS1 MOOD1 subset of safe MOODS2\{73l\}
   and safe MOODS2 subset of safe MOODS1 MOOD1\{73l,m\}.

4.8.1.

E) PROP :: DEC ; LAB.
EA) FIELDSETY :: FIELDS ; EMPTY.
F) QUALITY :: MODE ; MOID TALLY ; DYADIC ; label.

a) QUALITY down TALLY up TALLY i STACK NEW FLAGETY
   PROPSETY1 TALLY QUALITY TAX PROPSETY2 defining INDICATOR with TAX:
   where NEST TALLY QUALITY TAX independent PROPSETY1\{71aa,ab,ac,ad,ae\}, TAX token.

b) QUALITY NEST applied INDICATOR with TAX:
   where NEST TALLY QUALITY TAX identified in NEST with FLAGETY\{72a\},
   where TAX acceptable to NEST with FLAGETY\{ba\}, TAX token,
   where <NEST> is <down TALLY up TALLY STACK>.

ba) WHETHER TAX acceptable to NEST with FLAGETY:
   unless NEST indicates sublanguage\{22aa,ab\} and <TAX> is <TAB>, WHETHER true;
   WHETHER TAX not protected in FLAGETY\{bb,bc\}.

bb) WHETHER TAB not protected in EMPTY: WHETHER true.

bc) WHETHER TAB not protected in FLAG:
   WHETHER <FLAG> is not <protected TAB>.

c) MODE field FIELDSETY1 MODE field TAG FIELDSETY2 defining field selector with TAG:
   where MODE field TAG independent FIELDSETY1\{71a\}, TAG token.

5.1.

D) PRIMARY ::
    slice\{532a\} coercee ; call\{543a\} coercee ; cast\{551a\} coercee ; denoter\{80a\} coercee;
    unless NEST indicates sublanguage\{22aa,ab\}, format text\{A341a\} coercee;
applied identifier with TAG{48b} coercee ; ENCLOSED clause{31a,33a,c,d,e,34a,35a}.

5.2.1.1.

a) REF to MODE DOWN UP STACK assignation:
   REF to DOWN up TALLY STACK destination{b}, becomes token,
   MODE down TALLY UP STACK source{c}.

c) MODE1 NEST source:
   unconditionally strong MODE2 NEST unit{32d},
   where MODE1 deflexes to MODE2{47a,b,c,-}.

5.2.2.1.

a) boolean DOWN UP STACK identity relation:
   where DOWN UP STACK indicates sublanguage{22aa,ab},
   where <MODE> is not <ROWS of MODE1>,
   soft reference to MODE DOWN up TALLY TERTIARY1{5B}, identity relator{b},
   unconditionally strong reference to MODE down TALLY UP STACK TERTIARY2{5B} ;
   unless DOWN UP STACK indicates sublanguage{22aa,ab},
   where soft balances SORT1 and SORT2{32f},
   SORT1 reference to MODE DOWN up TALLY STACK TERTIARY1{5B}, identity relator{b},
   SORT2 reference to MODE down TALLY UP STACK TERTIARY2{5B}.

5.2.3.1.

a) reference to MODE NEST LEAP generator:
   where NEST indicates sublanguage{22aa,ab}
   implies NEST LEAP generator acceptable{aa,-},
   LEAP token, actual MODE NEST new found declarer{46a}.

aa) WHETHER NEST LAYER local generator acceptable:
   where <LAYER> is <new nonlocal FLAGETY PROPSETY>,
   WHETHER NEST local generator acceptable{aa} ;
   where <LAYER> is <new local FLAGETY DEC DECSETY LABSETY>,
   WHETHER NEST local generator occurs after DEC{ab} ;
   where <LAYER> is <new found FLAGETY>, WHETHER false.

ab) WHETHER down TALLY1 UP STACK local generator occurs after TALLY2 QUALITY TAX:
   WHETHER TALLY1 greater than TALLY2{ac,-}.

ac) WHETHER TALLY1 TALLY2 greater than TALLY1 : WHETHER true.

b) reference to MODINE NEST LEAP sample generator:
   where NEST indicates sublanguage{22aa,ab} implies <LEAP> is <local>, LEAP token,
   actual MODINE NEST new found declarer{44b,46a} ;
   where <LEAP> is <local>, actual MODINE NEST new found declarer{44b,46a}.

5.2.4.1.

a) UNCSTR reference to MODE NEST nihil:
   where NEST indicates sublanguage{22aa,ab} implies <UNCSTR> is <unconditionally strong>,
   nil token.

5.3.2.1.
AA) NUPWER :: lower ; upper ; new lower.

a) REFETY MODE1 DOWN UP STACK slice :
   weak REFLEXETY.ROWS1 of MODE1 DOWN up TALLY STACK PRIMARY{SD},
   ROWS1 leaving EMPTY down TALLY UP STACK indexer{b,c,-} STYLE bracket,
   where <REFETY> is derived from <REFLEXETY>{531b,c,-};
   where <MODE1> is <ROWS2 of MODE2>,
   weak REFLEXETY.ROWS1 of MODE2 DOWN up TALLY STACK PRIMARY{SD},
   ROWS1 leaving ROWS2 down TALLY UP STACK indexer{b,d,-} STYLE bracket,
   where <REFETY> is derived from <REFLEXETY>{531b,c,-}.

b) row ROWS leaving ROWSETY1 ROWSETY2 DOWN UP STACK indexer :
   row leaving ROWSETY1 DOWN up TALLY STACK indexer{c,d,-}, and also token,
   ROWS leaving ROWSETY2 down TALLY UP STACK indexer{b,c,d,-}.

d) row leaving row NEST indexer :
   NEST trimmer[f];
   optional NEST new lower bound[da].

da) optional NEST NUPWER bound :
   NEST NUPWER bound{g,46m,n};
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

f) DOWN UP STACK trimmer :
   optional DOWN up TALLY STACK lower bound[da], up to token,
   optional down TALLY1 up TALLY2 upper bound[da],
   optional down TALLY2 UP STACK new lower bound[da].

5.4.1.1.

a) procedure yielding MOID NEST1 routine text :
   formal MOID NEST1 declarer{46b}, routine token,
   unconditionally strong MOID NEST1 new local unit[32d].

b) procedure with PARAMETERS yielding MOID DOWN UP STACK1 routine text :
   DOWN up TALLY STACK1 new local DECS2 declarative
   defining new local DECS2{e} brief pack,
   where DECS2 like PARAMETERS{c,d,-},
   formal MOID down TALLY up TALLY STACK1 declarer{46b}, routine token,
   unconditionally strong MOID down TALLY UP STACK1 new local DECS2 unit[32d].

e) DOWN UP STACK2 declarative defining NEW DECS2 :
   formal DOWN up TALLY STACK2 declarer{46b},
   down TALLY UP STACK2 MODE parameter joined definition of DECS2{1b,c};
   where <DECS2> is <DECS3 DECS4>, formal MODE DOWN up TALLY1 STACK2 declarer{46b},
   down TALLY1 up TALLY2 STACK2 MODE parameter joined definition of DECS3{1b,c},
   and also token, down TALLY2 UP STACK2 declarative defining NEW DECS4{e}.

5.4.2.1.

a) MOID DOWN UP STACK DYADIC formula :
   MODE1 DOWN up TALLY1 STACK new found DYADIC TALLETY operand{c,-},
   procedure with MODE1 parameter MODE2 parameter yielding MOID
   down TALLY1 up TALLY1 STACK applied operator with TAD{48b},
   where DOWN UP STACK TALLY1 DYADIC TAD
   identified in DOWN UP STACK with FLAGETY{72a},
MODE2 down TALLY1 UP STACK new found DYADIC TALLY operand[c,-].

b) MOID NEST MONADIC formula :
   procedure with MODE parameter yielding MOID NEST applied operator with TAM[48b],
   MODE NEST new found MONADIC operand[c].

5.4.3.1.

a) MOID DOWN UP STACK call :
   meek procedure with PARAMETERS yielding MOID DOWN up TALLY STACK PRIMARY[5D],
   actual down TALLY UP STACK new local PARAMETERS brief pack[b,c].

b) actual DOWN UP STACK PARAMETERS PARAMETER :
   actual DOWN up TALLY STACK PARAMETERS[b,c], and also token,
   actual down TALLY UP STACK PARAMETER[c].

c) actual NEST MODE parameter :
   unconditionally strong MODE NEST unit[32d].

5.4.4.1.

a) UNCSTR MOID NEST jump :
   where NEST indicates sublanguage[22aa,ab], where <UNCSTR> is <unconditionally strong>,
   go to token, label NEST applied identifier with TAG[48b];
   unless NEST indicates sublanguage[22aa,ab], go to[b] option,
   label NEST applied identifier with TAG[48b].

5.5.1.1.

a) MOID DOWN UP STACK cast :
   formal MOID DOWN up TALLY STACK declarer[46b],
   unconditionally strong MOID down TALLY UP STACK
   ENCLOSED clause{31a,33a,c,d,e,34a,35a,-}.

5.5.2.1.

a) UNCSTR MOID NEST skip :
   where NEST indicates sublanguage[22aa,ab] implies <UNCSTR> is <unconditionally strong>,
   skip token.

6.1.1.

a) UNCSTR MOID FORM coercee :
   where <FORM> is <MORF>, STRONG[A] MOID FORM ;
   where <FORM> is <COMORF>, STRONG[A] MOID COMORF,
   unless <STRONG MOID> is <deprocedured to void>.

6.5.1.

AA) STRING :: structured with flexible row of character field letter aleph digit one mode.

d) widened to row of character FORM :
   MEEK{61C} BYTES FORM ; MEEK{61C} STRING FORM.
da) widened to STRING FORM : MEEK{61C} character FORM.

6.6.1.

a) rowed to REFETY ROWS1 of MODE FORM :
where NEST indicates sublanguage{22aa,ab} implies <REFETY> is <EMPTY>,
where <ROWS1> is <row>, STRONG{61A} REFLEXETY MODE FORM,
where <REFETY> is derived from <REFLEXETY>{531b,c,-} ;
where NEST indicates sublanguage{22aa,ab} implies <REFETY> is <EMPTY>,
where <ROWS1> is <row ROWS2>, STRONG{61A} REFLEXETY ROWS2 of MODE FORM,
where <REFETY> is derived from <REFLEXETY>{531b,c,-}.

7.1.1.

C) PREFSETY :: PREF PREFSETY ; EMPTY.
CA) MABEL :: MODE ; label.
CB) PRADIC :: PRAM ; DYADIC.

a) WHETHER FIELD independent FIELDSETY :
WHETHER FIELD unrelated to FIELDSETY{b,ba,bb}.

aa) WHETHER NEST TALLY MABEL TAG independent PROPSETY :
WHETHER NEST TALLY MABEL TAG unrelated to PROPSETY{bc,bd,c}.

ab) WHETHER NEST TALLY DYADIC TAD independent PROPSETY :
where NEST indicates sublanguage{22aa,ab},
where PROPSETY1 collected properties from NEST{af,ag},
WHETHER NEST TALLY DYADIC TAD unrelated to PROPSETY1 PROPSETY{bc,bd,c} and DYADIC TAD acceptable caption in PROPSETY1 PROPSETY{al} ;
unless NEST indicates sublanguage{22aa,ab},
WHETHER NEST TALLY DYADIC TAD unrelated to PROPSETY{bc,bd,c}.

ac) WHETHER NEST TALLY DUO TAD independent PROPSETY :
where NEST indicates sublanguage{22aa,ab},
where PROPSETY1 collected properties from NEST{af,ag},
WHETHER NEST TALLY DUO TAD unrelated to PROPSETY1 PROPSETY{bc,bd,c} and TALLY TAD follows priority declaration in PROPSETY1 PROPSETY{ah,ai} ;
unless NEST indicates sublanguage{22aa,ab},
WHETHER NEST TALLY DUO TAD unrelated to PROPSETY{bc,bd,c}.

ad) WHETHER NEST TALLY MONO TAM independent PROPSETY :
where NEST indicates sublanguage{22aa,ab},
where PROPSETY1 collected properties from NEST{af,ag},
WHETHER NEST TALLY MONO TAM unrelated to PROPSETY1 PROPSETY{bc,bd,c} and TAM acceptable caption in PROPSETY1 PROPSETY{ap} ;
unless NEST indicates sublanguage{22aa,ab},
WHETHER NEST TALLY MONO TAM unrelated to PROPSETY{bc,bd,c}.

ae) WHETHER NEST TALLY1 MOID TALLY2 TAB independent PROPSETY :
where NEST indicates sublanguage{22aa,ab},
where PROPSETY1 collected properties from NEST{af,ag},
WHETHER NEST TALLY1 MOID TALLY2 TAB unrelated to PROPSETY{bc,bd,c} and TAB not operator in PROPSETY1{aj,ak} ;
unless NEST indicates sublanguage{22aa,ab},
WHETHER NEST TALLY1 MOID TALLY2 TAB unrelated to PROPSETY{bc,bd,c}.
af) where PROPSETY collected properties from NEW FLAGETY:
   where <PROPSETY> is <EMPTY>.
ag) where PROPSETY1 PROPSETY2 collected properties from NEST NEW FLAGETY PROPSETY3:
   where <PROPSETY2> is <PROPSETY3> and PROPSETY1 collected properties from NEST[af].
ah) WHETHER TALLY1 TAD follows priority declaration in EMPTY: WHETHER false.
ai) WHETHER TALLY1 TAD follows priority declaration in PROPSETY PROP:
   where <PROP> is <TALLY2 DYADIC TAD>,
   WHETHER TALLY1 greater than TALLY2{523ac,-};
   unless <PROP> is <TALLY2 DYADIC TAD>,
   WHETHER TALLY1 TAD follows priority declaration in PROPSETY{ah,ai}.
aj) WHETHER TAB not operator in EMPTY: WHETHER true.
ak) WHETHER TAB not operator in PROPSETY PROP:
   where <PROP> is <TALLY PRADIC TAB>, WHETHER false;
   unless <PROP> is <TALLY PRADIC TAB>,
   WHETHER TAB not operator in PROPSETY{aj,ak}.
al) WHETHER DYADIC TAD acceptable caption in PROPSETY:
   where <TAD> is <bold TAG>, WHETHER true;
   unless <TAD> is <bold TAG>, WHETHER TAD used dyadically in PROPSETY{am,an,ao,-}.
am) WHETHER TAD used dyadically in EMPTY: WHETHER false.
an) WHETHER TAD used dyadically in TALLY DYADIC TAD: WHETHER true.
ao) WHETHER TAD used dyadically in PROPS PROP:
   WHETHER TAD used dyadically in PROPS{an,ao,-} or TAD used dyadically in PROP{an,-}.
ap) WHETHER TAM acceptable caption in PROPSETY:
   where <TAM> is <bold TAG>, WHETHER true;
   unless <TAM> is <bold TAG>, WHETHER TAM used monadically in PROPSETY{aq,ar,as,-}.
aq) WHETHER TAM used monadically in EMPTY: WHETHER false.
ar) WHETHER TAM used monadically in TALLY MONO TAM:
   WHETHER true.
as) WHETHER TAM used monadically in PROPS PROP:
   WHETHER TAM used monadically in PROPS{ar,as,-} or TAM used monadically in PROP{ar,-}.
b) WHETHER FIELD unrelated to EMPTY: WHETHER true.
ba) WHETHER FIELD1 unrelated to FIELD2 FIELDS2:
   WHETHER FIELD1 unrelated to FIELD2{bb} and FIELD1 unrelated to FIELDS2{ba,bb}.
bb) WHETHER MODE1 field TAG1 unrelated to MODE2 field TAG2:
   WHETHER <TAG1> is not <TAG2>.
bc) WHETHER NEST PROP unrelated to EMPTY: WHETHER true.
bd) WHETHER NEST PROP1 unrelated to PROP2 PROPS2:
   WHETHER NEST PROP1 unrelated to PROP2{c} and NEST PROP1 unrelated to PROPS2{bd,c}.
c) WHETHER NEST TALLY1 QUALITY1 TAX1 unrelated to TALLY2 QUALITY2 TAX2:
   unless <TAX1> is <TAX2>, WHETHER true;
   where <TAX1> is <TAX2> and <TAX1> is <TAXO>,
   WHETHER NEST QUALITY1 independent QUALITY2{d}.
d) WHETHER NEST QUALITY1 independent QUALITY2:
   where NEST QUALITY1 related QUALITY2{e,f,g,h,i,j,-}, WHETHER false;
   unless NEST QUALITY1 related QUALITY2{e,f,g,h,i,j,-}, WHETHER true.
e) WHETHER NEST MONO related DUO: WHETHER false.
f) WHETHER NEST DUO related MONO: WHETHER false.
g) WHETHER NEST PRAM related DYADIC: WHETHER false.
h) WHETHER NEST DYADIC related PRAM: WHETHER false.
i) WHETHER NEST procedure with MODE1 parameter MODE2 parameter yielding MOID1 related
procedure with MODE3 parameter MODE4 parameter yielding MOID2:
where NEST indicates sublanguage{22aa,ab},
WHETHER MODE1 meekly related MODE3{na} and MODE2 meekly related MODE4{na} ;
unless NEST indicates sublanguage{22aa,ab},
WHETHER MODE1 firmly related MODE3{k} and MODE2 firmly related MODE4{k}.

j) WHETHER NEST procedure with MODE1 parameter yielding MOID1 related
procedure with MODE2 parameter yielding MOID2:
where NEST indicates sublanguage{22aa,ab},
WHETHER MODE1 meekly related MODE2{na} ;
unless NEST indicates sublanguage{22aa,ab}, WHETHER MODE1 firmly related MODE2{k}.

na) WHETHER MODE1 meekly related MODE2:
WHETHER MODE1 deprefs to NONPREF{nb} and MODE2 deprefs to NONPREF{nb}.

nb) WHETHER MODE deprefs to NONPREF2:
where <MODE> is <PREFSETY NONPREF1>,
WHETHER NONPREF1 equivalent NONPREF2{73a}.

7.2.1.

AA) LSPLAIN :: LSREAL ; LSINT ; character ; boolean.
AB) SPECIAL STRUCTURE :: LSBITS ; LSBYTES ; STRING ; FILE ; CHANNEL.
AC) SPECIAL PARAMETER ::
   ROWSTYPE ; NUMBERTYPE ; PUTTYPE ; GETTYPE ; PUTBINTYPE ; GETBINTYPE.
AD) LSREAL ::
   A number of protonotions (terminal productions of 'MODE'), each of which is equivalent to
   a different mode, selected from those specified by the actual-declarers of the mode
   declarations of L real (Report 10.2.2.d), where L stands for some sequence (possibly null)
   of long or short.
AE) LSINT ::
   A number of protonotions (terminal productions of 'MODE'), each of which is equivalent to
   a different mode, selected from those specified by the actual-declarers of the mode
   declarations of L int (Report 10.2.2.c), where L stands for some sequence (possibly null)
   of long or short.
AF) LSBITS ::
   A number of protonotions (terminal productions of 'MODE'), each of which is equivalent to
   a different mode, selected from those specified by the actual-declarers of the mode
   declarations of L bits (Report 10.2.2.g), where L stands for some sequence (possibly null)
   of long or short.
AG) LSBYTES ::
   A number of protonotions (terminal productions of 'MODE'), each of which is equivalent to
   a different mode, selected from those specified by the actual-declarers of the mode
   declarations of L bytes (Report 10.2.2.h), where L stands for some sequence (possibly null)
   of long or short.
AH) FILE ::
   A protonotion (a terminal production of 'MODE'), which is equivalent to that specified
   by the actual-declarer of the mode declaration of file (Report 10.3.1.3.a).
AI) CHANNEL ::
   A protonotion (a terminal production of 'MODE'), which is equivalent to that specified
   by the actual-declarer of the mode declaration of channel (Report 10.3.1.2.a).
AJ) ROWSTYPE ::
   A protonotion (a terminal production of 'MODE'), which is equivalent to that specified
by the actual-declarer of the mode declaration of `rows' (Report 10.2.3.1.a).

AK) NUMBERTYPE ::
A protonotion (a terminal production of 'MODE'), which is equivalent to specified
by the actual-declarer of the mode declaration of `number' (Report 10.3.2.1.a).

AL) PUTTYPE ::
A protonotion (a terminal production of 'MODE'), which is equivalent to that specified
by the declarer [ ] union ( outtype, proc ( ref file ) void ), see the Report 10.3.2.2.b.

AM) GETTYPE ::
A protonotion (a terminal production of 'MODE'), which is equivalent to that specified
by the declarer [ ] union ( inttype, proc ( ref file ) void ), see the Report 10.3.2.2.d.

AN) PUTBINTYPE ::
A protonotion (a terminal production of 'MODE'), which is equivalent to that specified
by the declarer [ ] outtype, see the Report 10.3.2.2.b.

AO) GETBINTYPE ::
A protonotion (a terminal production of 'MODE'), which is equivalent to that specified
by the declarer [ ] inttype, see the Report 10.3.2.2.d.

a) WHETHER NEST1 PROP identified in NEST2 NEW FLAGETY1 PROPSETY with FLAGETY2 :
where NEST1 PROP resides in PROPSETY{b,c,-}, where <FLAGETY2> is <FLAGETY1>,
WHERE NEST1 PROP acceptable from NEST2 NEW FLAGETY1{cc} ;
where NEST1 PROP independent PROPSETY{71aa,ab,ac,ad,ae},
WHERE NEST1 PROP identified in NEST2 with FLAGETY2{a,-}.

b) WHETHER NEST PROP1 resides in PROPS2 PROP2 :
WHETHER NEST PROP1 resides in PROP2{c,-} or NEST PROP1 resides in PROPS2{b,c,-}.

ba) WHETHER FIELD1 resides in FIELDS2 FIELD2 :
WHETHER FIELD1 resides in FIELD2{ca,-} or FIELD1 resides in FIELDS2{ba,ca,-}.

c) WHETHER NEST TALLY1 QUALITY1 TAX resides in TALLY2 QUALITY2 TAX :
where <QUALITY1> is <label>, WHETHER <QUALITY2> is <label> ;
where <QUALITY1> is <DYADIC> and <QUALITY2> is <DYADIC>,
WHERE NEST application TALLY1 satisfies definition TALLY2{cb} ;
where <QUALITY1> is <MOID1 TALLETY> and <QUALITY2> is <MOID2 TALLETY>,
WHERE MOID1 equivalent MOID2{73a} and NEST application TALLY1 satisfies definition TALLY2{cb}.  

c a) WHETHER MODE field TAG resides in MODE field TAG : WHETHER true.

cb) WHETHER NEST application TALLY1 satisfies definition TALLY2 :
where NEST indicates sublanguage{22aa,ab},
WHERE TALLY1 greater than TALLY2{5231ac,-} ; unless NEST indicates sublanguage{22aa,ab}, WHETHER true.

cc) WHETHER NEST1 TALLY QUALITY TAX acceptable from NEST2 :
where NEST2 indicates sublanguage{22aa,ab}, WHETHER true ; unless NEST1 indicates sublanguage{22aa,ab} or NEST2 indicates sublanguage{22aa,ab},
WHERE true ;
where NEST1 indicates sublanguage{22aa,ab}, unless NEST2 indicates sublanguage{22aa,ab},
WHERE QUALITY acceptable{cd} and QUALITY TAX visible through window{ck}.

d) WHETHER QUALITY acceptable :
where <QUALITY> is <label> or <QUALITY> is <DYADIC>, WHETHER true ;
where <QUALITY> is <MOID TALLETY>, WHETHER MOID suitable{ce}.

c e) WHETHER MOID suitable :
where <MOID> is <void> or <MOID> is <LSPLAIN>, WHETHER true ;
where <MOID> is <reference to MODE>, WHETHER MODE suitable{ce} ;
where <MOID> is <procedure PARAMETY yielding MOID1>,
Whether MOID1 suitable{ce} and PARAMETY suitable{cf, cg, ch};
where <MOID> equivalent <SPECIAL STRUCTURE>{73a}, WHETHER true;
where <MOID> is <structured with FIELDS mode>,
unless <MOID> equivalent <SPECIAL STRUCTURE>{73a}, WHETHER FIELDS suitable{ci, cj};
where <MOID> is <ROWS of MODE>,
Whether MODE suitable and <MODE> is not <ROWS1 of MODE1>.

cf) WHETHER EMPTY suitable : WHETHER true.

cg) WHETHER with MODE parameter suitable :
  Whether MODE suitable{ce} or <MODE> equivalent <SPECIAL PARAMETER>{73a}.

ch) WHETHER with PARAMETER PARAMETERS suitable :
  Whether with PARAMETER suitable{cg} and with PARAMETERS suitable{cg, ch}.

ci) WHETHER MODE field TAG suitable :
  Whether MODE suitable{ce} and <MODE> is not <ROWS of MODE1>.

cj) WHETHER FIELD FIELDS suitable :
  WHETHER FIELD suitable{ci} and FIELDS suitable{ci, cj}.

ck) WHETHER QUALITY TAX visible through window :
  This hypernotion produces either 'WHETHER true' or 'WHETHER false', depending upon the particular 'QUALITY TAX'. Which one it produces may be deduced from Section 4 above.

8.0.1.

a) MOID NEST denoter :
  Where NEST indicates sublanguage{22aa, ab},
  where <MOID> is <LSPLAIN> or <MOID> is <LSBITS> or <MOID> is <STRING>,
  fragment{92a} sequence option,
  MOID denotation{810a, 811a, 812a, 813a, 814a, 82a, b, c, 83aa, -};
  Unless NEST indicates sublanguage{22aa, ab}, unless <MOID> is <STRING>,
  fragment{92a} sequence option,
  MOID denotation{810a, 811a, 812a, 813a, 814a, 815a, 82a, b, c, 83aa, -}.

8.3.1.

a) row of character denotation :
  quote symbol, string{b} option, quote symbol.

aa) STRING denotation :
  quote symbol, string{b} option, quote symbol.

c) * string denotation : row of character denotation{a} ; STRING denotation{aa}.

10.1.1.

a) program text :
  STYLE begin token, down i up TALLY new local LAYER1 preludes{b}, parallel token,
  down TALLY UP new local LAYER1 tasks{d} PACK, STYLE end token.

b) DOWN UP STACK1 preludes :
  DOWN up TALLY1 STACK1 standard prelude with DECS1{c},
  down TALLY1 up TALLY2 STACK1 library prelude with DECSETY2{c},
  down TALLY2 UP STACK1 system prelude with DECSETY3{c},
  where <STACK1> is <new local EMPTY new local DECS1 DECSETY2 DECSETY3>.
c) NEST1 EXTERNAL prelude with DECSETY1:
   unconditionally strong void NEST1 series with DECSETY1{32b}, go on token;
   where <DECSETY1> is <EMPTY>, EMPTY,
   where <NEST1> is <down TALLY up TALLY STACK>.

d) DOWN UP STACK1 tasks:
   DOWN up TALLY system task{e} list, and also token,
   down TALLY UP STACK1 user task{f} list.

  e) NEST1 system task: unconditionally strong void NEST1 unit{32d}.

f) DOWN UP STACK1 user task:
   DOWN up TALLY1 STACK2 particular prelude with DECS{c},
   down TALLY1 up TALLY2 STACK2 particular program{g} PACK,
   go on token, down TALLY2 UP STACK2 particular postlude{i},
   where <STACK2> is <STACK1 new local DECS STOP>.

g) DOWN UP STACK2 particular program:
   DOWN up TALLY STACK2 new local LABSETY3 joined label definition of LABSETY3{h},
   unconditionally strong void down TALLY UP STACK2
   new local LABSETY3 ENCLOSED clause{31a,33a,34a,35a} ;
   DOWN up TALLY STACK2 new local sublanguage LABSETY3
   joined label definition of LABSETY3{h},
   unconditionally strong void down TALLY UP STACK2
   new local sublanguage LABSETY3 ENCLOSED clause{31a,33a,34a,35a}.

h) DOWN UP STACK joined label definition of LABSETY:
   where <LABSETY> is <EMPTY>, EMPTY, where <DOWN UP> is <down TALLY up TALLY> ;
   where <LABSETY> is <LAB1 LABSETY1>,
   DOWN up TALLY STACK label definition of LAB1{32c},
   down TALLY UP STACK joined label definition of LABSETY1{i}.

i) NEST2 particular postlude:
   unconditionally strong void NEST2 series with STOP{32b}.

10.3.4.1.1.

b) DOWN UP STACK collection:
   fragment{92a} sequence option, DOWN UP STACK picture{c};
   fragment{92a} sequence option, DOWN up TALLY1 STACK insertion{d},
   down TALLY1 up TALLY2 STACK replicator{g},
   down TALLY2 up TALLY3 STACK collection{b} list brief pack,
   fragment{92a} sequence option, down TALLY3 UP STACK insertion{d}.

c) DOWN UP STACK picture:
   optional DOWN up TALLY STACK TYPE pattern{ca}, down TALLY UP STACK insertion{d}.

c a) optional NEST TYPE pattern:
   NEST TYPE pattern{A342a,A343a,A344a,A345a,A346a,A347a,A348a,b,A349a,A34Aa} ;
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

d) DOWN UP STACK insertion:
   optional DOWN up TALLY STACK literal{da},
   optional sequence of down TALLY UP STACK alignment{db}.

da a) optional NEST UNSUPPRESSETY literal:
   NEST UNSUPPRESSETY literal{i} ;
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

d b) optional sequence of NEST alignment:
   NEST alignment{e} sequence ;
EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

e) DOWN UP STACK alignment:
   DOWN up TALLY STACK replicator{g}, alignment code{f},
   optional down TALLY UP STACK literal{da}.

g) NEST replicator:
   optional NEST unsuppressible replicator{ga}.

ga) optional NEST unsuppressible replicator:
   NEST unsuppressible replicator{h};
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

i) DOWN UP STACK UNSUPPRESSETY literal:
   DOWN up TALLY1 STACK UNSUPPRESSETY replicator{g,h},
   unconditionally strong row of character
   down TALLY1 up TALLY2 STACK denoter{80a} coercee{61a},
   optional down TALLY2 UP STACK unsuppressible literal{da}.

k) DOWN UP STACK UNSUPPRESSETY COMARK frame:
   DOWN up TALLY STACK insertion{d}, down TALLY UP STACK replicator{g},
   UNSUPPRESSETY suppression{I}, COMARK marker{A342d,f,A346b}.

10.3.4.2.1.

a) DOWN UP STACK integral pattern:
   optional DOWN up TALLY STACK sign mould{aa}, down TALLY UP STACK integral mould{b}.

aa) optional NEST sign mould:
   NEST sign mould{c};
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

c) DOWN UP STACK sign mould:
   optional sequence of DOWN up TALLY STACK unsuppressible zero frame{ca},
   down TALLY UP STACK unsuppressible sign frame{A341j}.

ca) optional sequence of NEST UNSUPPRESSETY COMARK frame:
   NEST UNSUPPRESSETY COMARK frame{A341k} sequence;
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

10.3.4.3.1.

a) DOWN UP STACK real pattern:
   optional DOWN up TALLY STACK sign mould{A342aa},
   down TALLY UP STACK variable point mould{b} or alternatively
   down TALLY UP STACK floating point mould{c}.

b) DOWN UP STACK variable point mould:
   DOWN up TALLY1 STACK integral mould{A342b},
   down TALLY1 up TALLY2 STACK point frame{A341j},
   optional down TALLY2 UP STACK integral mould{ba}.

ba) optional NEST integral mould:
   NEST integral mould{A342b};
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

c) DOWN UP STACK floating point mould:
   DOWN up TALLY1 STACK variable point mould{b} or alternatively
   DOWN up TALLY1 STACK integral mould{A342b},
   down TALLY1 up TALLY2 STACK exponent frame{A341j},
   down TALLY2 UP STACK integral pattern{A342a}.
10.3.4.5.1.

a) DOWN UP STACK complex pattern:
   DOWN up TALLY1 STACK real pattern{A343a},
   down TALLY1 up TALLY2 STACK complex frame{A341j},
   down TALLY2 UP STACK real pattern{A343a}.

10.3.4.7.1.

a) DOWN UP STACK bits pattern:
   DOWN up TALLY STACK RADIX frame{b}, down TALLY UP STACK integral mould{A342b}.

10.3.4.8.1.

a) DOWN UP STACK integral choice pattern:
   DOWN up TALLY STACK insertion{A342b}, letter c symbol,
   down TALLY UP STACK literal{A341i} list brief pack.

b) DOWN UP STACK boolean choice pattern:
   DOWN up TALLY1 STACK insertion{A341d}, boolean marker{A344b}, open token,
   down TALLY1 up TALLY2 STACK literal{A341i}, and also token,
   down TALLY2 UP STACK literal{A341i}, close token.

10.3.4.9.1.

a) DOWN UP STACK format pattern:
   DOWN up TALLY STACK insertion{A341d}, letter f symbol,
   meek FORMAT down TALLY UP STACK ENCLOSED clause{31a,34a}.

10.3.4.10.1.

a) DOWN UP STACK general pattern:
   DOWN up TALLY STACK insertion{A341d}, letter g symbol,
   optional briefly packed down TALLY UP STACK width specification{aa}.

aa) optional briefly packed NEST width specification:
   NEST width specification{b} brief pack;
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

b) DOWN UP STACK width specification:
   meek integral DOWN up TALLY STACK unit{32d},
   optional down TALLY UP STACK after specification{ba}.

ba) optional NEST after specification:
   NEST after specification{b};
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>.

c) DOWN UP STACK after specification:
   and also token, meek integral DOWN up TALLY STACK unit{32d},
   optional down TALLY UP STACK exponent specification{ca}.

ca) optional NEST exponent specification:
   NEST exponent specification{d};
   EMPTY, where <NEST> is <down TALLY up TALLY STACK>. 