Syntax	Meaning	Examples	Notes

```
1 module ::= DIRECTORY (nd (: TYPE  (nt | nd) ) ( (TYPE nt | TYPE nd) | TYPE nd), ... ;
    ?(USING [ (nu, RESTRICT[nd), [$nu, ; ... ] ] ), ...
           ( interface | implementation )
2 interface ::= nm, !.. : ?CEDAR DEFINITIONS     [(niv | nit):MKINSTANCETYPE[nit]), ...]=>
  ?locks ?(IMPORTS ( (niv : | ) nit ), ...)              [nm: TYPE nm] IN
  ?•(SHARES ns, ...)                      -- Access to PRIVATE names in ns is allowed.
  ~ ?•access12 { ?open7 (d | b); !.. }    LET REC nm~open [ ?(lock(~locks, ) (d | b), ... ]  IN nm
3 implementation ::= nm : ?CEDAR          l [( ( niv | nit ):MKINSTANCETYPE[nit]), ...]=>
  ?safety ( PROGRAM ?drType42  |              [(ne: ne)  , ... , nm: TYPE nm , CONTROL: PROGRAM] IN
         MONITOR ?drType42  ( ( || (LOCKS LOCK:MONITORLOCK IN LET l(IN LOCK) IN | ))
  ?(IMPORTS ( (niv : | ) nit ), ...)  LET b(~NEWPROGINSTANCE[block].UNCONS IN
  ?(EXPORTS ne, ...)                          [ (ne~BINDDFROM[ne, b() ), ... , nm~b(, CONTROL~b( nm ]
  ?•(SHARES ns, ...)                  where the body of the block is desugared thus:
  ~ ?•access12 block .                        [( | ( | l(~locks,)) (d | b), ... , nm: PROGRAM drType~{s;
  ...}]
4 safety ::= SAFE | UNSAFE  --In 3, 41.
5 locks ::= LOCKS e ?( USING nu: t)  l ?( [nu : t] ) IN e
```

```
DIRECTORY
  Rope: TYPE USING [ROPE, Compare],
  CIFS: TYPE USING [OpenFile,Error,Open,read],
  IO: TYPE IOStream,
  Buffer: TYPE;
BufferImpl: MONITOR [f: CIFS.OpenFile]
  LOCKS Buffer.GetLock[h]^
  USING h: Buffer.Handle
  IMPORTS Files: CIFS, IO, Rope
  EXPORTS Buffer
  ~ { -- module body -- } .
```

```
-- There should always be a USING ...
-- unless most of the interf...
--   or it is exported.
-- Implementations can have ar...
-- LOCKS only in MONITOR, to spe...
--   a non-standard lock.
-- Note the absence of semicol...
-- EXPORTS in PROGRAM or MONITOR.
-- Note the final dot.
```

```
6 block ::=
  ?(CHECKED | UNCHECKED | TRUSTED)
  { ?open ?enable ?body               open LET n((, ... : EXCEPTION~NEWLABEL[] , ...
      ?(EXITS (n, !..=>s); ...) }       IN body enable BUT { (n((, ... => s ); ... }
  --In 3, 13, 15.                     -- But n(( is not visible in s.
7 open ::= OPEN ( n ~~ e | e ), !.. ;  LET n~lopen IN e.DEREF |  --The IN before !.. is a separator.
  In 2, 5, 17. •The ~~ may be written as :  LET BINDP[(e(DEREF).P,
                                        OPENPROCS[D(e.DEREF).P, l IN e.DEREF] ] ) IN !.. IN
8 enable::=ENABLE(enChoice|           BUT ( { enChoice|
  {enChoice; ...});                   { enChoice ... } )
  In 5, 17.
9 enChoice ::=( e, !.. | ANY ) => s   ( e | ANY ), ... => { s; REJECT; EXITS
  In 7, 27.1.                             Retry(=>GOTO Retry((14; Cont(=>GOTO Cont((14 }
10 body  ::= (d | b); !.. ; s; ...    LET; NEWFRAME[ REC [(d | b), ...] ].UNCONS IN { s; }
  In 5, 17.
```

```
CHECKED {
 OPEN Buffer, Rope;
 ENABLE  Buffer.Overflow=>GOTO HandleOvfl;
 stream: IO.Stream~IO.CreateFileStream["X"];
 x: INT_7;
 {OPEN b: GetBuffer[stream];
   ENABLE  {
   Files.Error--[error, file]--=>{
    stream.Put[IO.rope[error]];
    ERROR Buffer.Error["Help"] };
   ANY=>{ x_12; GOTO AfterQuit } };
  y: INT_9; ... };
 x_stream.GetInt; ...
EXITS
  AfterQuit=>{...};
  HandleOvfl=>{...} };
```

```
-- Unnamed OPEN OK for exported
--   interface or one with a U...
-- A single choice needn't be ...
-- Use a binding if a name's v...
-- Better to initialize declar...
-- A statement may be a nested ...
-- Multiple enable choices mus...
-- ERRORS can have parameters.
-- Choices are separated by se...
-- ANY must be last. ENABLE end...
-- Other bindings, decls and s...
-- Other statements in the out...
-- Multiple EXIT choices are no...
-- AfterQuit, HandleOvfl decla...
--    legal only in a GOTO in th...
```

```
11 declaration ::= n, !.. : ?access12 (varTC varTC ), ...
  In 2, 10, 43. VAR, READONLY only for interface var.
12 access ::= PUBLIC | PRIVATE
  In 2, 3, 11, 13, 50, 51, 53.
13 binding ::= n, !.. : ?access12 t ~  LET x( : t ~ ( -- The desugaring for n is at the end
  e |                                   e |
  t2  -- if t=TYPE  |                   t -- Same as e except for conflicting syntax. |
  CODE |                                NEWEXCEPTIONCODE[]  --tgSIGNAL or ERROR |
  ?INLINE ?(ENTRY | INTERNAL)  block6   l [d(: t.DOMAIN] IN LET r(~NEWFRAME[t.RANGE].UNCONS
                                          INET(r( IN {t.DOMAIN~d(; block; RETURN}
                                               BUT {Return((=>r()}
  „, ?TRUSTED MACHINE CODE {(e, ...);    MACHINECODE[(BYTESTOINSTRUCTION[e, ...]), ...]
  )                                     ) IN n, ... ~ x( -- e is evaluated only once.
  In 2, 10. •The ~ may be written as =.
  Block or MACHINE CODE only for proc types.
  •ENTRY and INTERNAL can also be before t.
```

```
HistValue: TYPE[ANY];
Histogram: TYPE~REF HistValue;
baseHist: READONLY Histogram;
AddHists: PROC[x, y: Histogram]
  RETURNS [Histogram];
LabelValue: PRIVATE TYPE~RECORD[
   first,last:INT,s:ROPE,x:REAL,f,g:INT,F:REF ANY];
Label: TYPE~REF LabelValue;
Next: PROC[l: Label] RETURNS[Label]~
  INLINE { RETURN [NARROW[l.r]] };
H: TYPE~Histogram11; Size: INT~10;
HistValue: PUBLIC TYPE~HV40.1;
baseHist: PUBLIC H_NEW[HistValue_ALL[17]];
x, y: HistValue_[ 20, 18, 16, 14, 12, 10, 8, 6, 4, 2, 0];
FatalError: ERROR[reason: ROPE]~CODE;
Setup: PROC [bh: Handle3, a: INT]~ENTRY {...};
i,j,k: INT_0; p,q: BOOL; lb: Label; main: Handle;
```

```
-- Interface: An exported type.
--         A type binding.
--         An exported variabl...
--         An exported proc.
--         PRIVATE only for  se...
--           stuff in an  inte...
--         An inline proc bind...
-- Implementation: Binds a TYPE...
--         PUBLIC for exports.
--         An exported variabl...
--         with initializati...
--         Binds an error.
--         Binds an entry proc...
```

```
14 Statement ::= sS                   { SIMPLELOOP {sS; GOTO Cont((; EXITS  Retry((=>NULL}
  In 6, 10, 17, 19.                       EXITS Cont((=>NULL }
15 sS ::=e1_e2 | e | block6 | escape | loop | TOVOID | e --must yield VOID-- | --all yield VOID--
16 escape ::= GOTO n | GO TO n |       EXORVAL[exception[code~ n((, args~NIL]] |
  EXIT | CONTINUE | •LOOP | •RETRY |  GOTO ( Exit(17 | Cont(9 | Loop(17| Retry(9) |
  (RETURN | RESUME) ?e |              { ?(r(13_e;) GOTO (Return(13 | Resume(13) } |
  •REJECT | ,„e _ STATE               THISEXCEPTION[]  | DUMPSTATE[e]
17 loop ::= (iterator | )             { ( iterator | done(~FALSE; Next(: PROC~{}; )
  (WHILE e | UNTIL e | )                { Test(~l IN (NOT e | e | FALSE);
  DO  ?•open7 ?•enable8 ?body10           {  OPEN SIMPLELOOP {
                                                IF Test([] OR done( THEN GOTO FINISHED;
                                                { enable body EXITS Loop(=>NULL }; Next([] }
  ?(REPEAT (n, !..=>s); ...) ENDLOOP      EXITS Exit(gNULL; (n, !..gs); ...; FINISHEDgNULL}}}
18 iterator ::= THROUGH  e |          FOR x(: e IN|e
            FOR (n : t | µ(n): t; | )   DECREASING TYPE~e, done(: BOOL_Range(.ISEMPTY;
                 ( ( | DECREASING) IN e |
                                        Next(: PROC~{ IF n ( >Range(.LAST | <Range(.FIRST )
                                          THEN done(_TRUE ELSE n_n.(SUCC | PRED) };
                                        n_Range(.(FIRST | LAST); |
                                      _ done(_e2BOOL~FALSE; Next(: PROC~{n_e2}; n_e1 ) ;
  e is a subrange. In FOR n: t ... , n is readonly except for the assignment in the iterator's desugaring.
```

```
AddHists[baseHist, baseHist]^;
Setup[bh~main, a~3];
{ENABLE FatalError=>RETURN[0]; []_f[3]; ... };
IF i>3 THEN RETURN[25] ELSE GOTO NotPresent;
FOR t:INT DECREASING IN [0..5) UNTIL f[t]>3 DO
  u: INT_0; ... ; u_t+4; ...
  REPEAT Out=>{...}; FINISHED=>{...} ENDLOOP;
THROUGH [1..5) DO i_i*i ENDLOOP;
FOR i: INT_1, i+2 WHILE i<8 DO j_j+i ... ;
FOR l: Label_lb, l.Next WHILE l#NIL DO ... ;
```

```
-- A statement can be an assig...
-- or an application without r...
-- or a block,
-- or an IF or an escape stater...
-- Do a loop. Try to declare t...
--   as shown. Avoid OPEN or EN...
--   after DO (use a block). FI...
--   must be last.
-- Raises i to the 6th power.
-- Accumulates odd numbers in ...
-- Sequences through a list of ...
```

```
19 expression ::= n | literal53 | application26 |
  (e | typeName37) . (9) n |
  prefixOp e | e1 infixOp e2  |       e . prefixOp e1 . infixOp[e2]
  e1 relOp (4) e2  |                  ( l [x(: De1, y( :De2] IN e1 relOp |
  e1 AND (2) e2  | e1 OR (1) e2  |    IF e1 THEN e2 ELSE FALSE | IF e1 THEN TRUE ELSE e2
  e ^ (9) | •STOP | ERROR |           e . DEREFERENCE | STOP[] | ERROR NAMELESSERROR |
  builtIn [ e1 ?( , e2, !..) ?applEn27]. | builtIn ?( [e2, ...] ) |
  funnyAppl e ?( [?argBinding27 ?applEn27] | funnyAppl ?( [argBinding ?applEn ] ) |
  [ argBinding27 ] |                  --Binding must coerce to a record, array, or •local string-- |
  s | subrange25 | if28 | select29 |  safeSelect32 | •withSelect34
  Precedence is in bold in rules 19-21.  All operators associate to the left except _, which associates
  to the right. Application has highest precedence. Subrange only after IN or THROUGH. s only in if 28 and select choices30 33 35.
20 prefixOp ::= @ (8) |   (7) | (~ | NOT) (3)   WORDPOINTER | UMINUS | NOT
21 infixOp ::= * | / | MOD (6) | + |   (5) | || (0)   TIMES | DIVIDE | REM | PLUS | MINUS | ASSIGN
22 relOp ::= ?NOT (  ?~ (= | < | >) | ?# ) |    ?NOT( ?NOT x(.(EQUAL | LESS | GREATER)[y(] | x(~=y( |
              (<=x(=y(| ( IN )(< | >) y( | x(>=y(_FIRST y(AND x(
                                         _ )BUT {BoundsFault=>FALSE} )
  --In 19, 30.
23 builtIn ::= -- These are enumerated in Table 4 5.
24 funnyAppl ::= FORK | JOIN | WAIT | NOTIFY | BROADCAST | SIGNAL | ERROR | RETURN WITH ERROR |
  •NEW | •START | •RESTART |„,TRANSFER WITH | „,RETURN WITH
25 subrange ::= (typeName37 | )       LET t(~(typeName | INT) , first(~( e1 | e1.SUCC) IN
        _ ) e1 .. e2 )( [] || ()        t(.MKSUBRANGE[first(, (e2 | e2.PRED )] BUT
  --In 19, 39, 48.                           {BoundsFault=>t(.MKEMPTYSUBRANGE[e1]}
```

```
lv: LabelValue13_[ i, 3, "Hello", 31.4E-1, f[i],
  g[x]+lb.f+j.PRED ];
p1: PROCESS RETURNS [INT]_FORK f[i, j];
ERROR NoSpace; WAIT bufferFilled;
RT: RTBasic.Type_CODE[LabelValue13];
h[ 3, NOT(i>j), i*j, i_3, i NOT >j, p OR q, lb^];
lv19_[first~0, last~10, x~3.14, g~2, f~5];
[first~i, last~j]_lv19;
BOOL_i IN [1..10]; FOR x: INT IN (0..11) DO ...;
b_( c IN Color54(red..green] OR x IN INT[0..10) );
```

```
-- A constructor with some sam...
--   expressions.
-- FunnyAppls take one unbrack...
--   arg; many return no resul...
--   must be statements.
-- An application with sample ...
-- Short for lv_LabelValue13[...
-- Assignment to VAR binding
--   (extractor).
-- Subrange only in types or w...
-- The INT is redundant.
```

Notation: item | item=choose one; ?item=zero or one item; terminals: SMALL CAPS, punctuation other than bold ()?| (parens are terminals only in rules 19, ...
item s ...=zero or more items, separated by s; item s !..=one or more items, separated by s; with s=";", a trailing ";" is opti...
Abbreviated non-terminals: b=binding13; d=declaration11; e=expression19; n=name56; s=statement14; t=type36. 4 Feb 83
Comments: •=obsolete; µ=efficiency hack; ,=unsafe; „=machine dependent; n56 means n is defined in rule 56; n2=n (the subscript is only for the desu... CLRM Summ...

	Syntax	Meaning	Examples	Notes
26	application ::= e [?argBinding ?applEn]	e, a(~[argBinding](. APPLY Za(?applEn)	fh_Files.Open[name~lb.s, mode~Files.read	Keywords are best for multi
27	argBinding ::= (n ~ (e | | µTRASH)), ... |	(n ~ (e | OMITTED | TRASH)), !.. |	! AccessDenied=>{...}; FatalError=>{...}];	-- Semicolons separate choices
	(e | | µTRASH), ...	(e | OMITTED | TRASH), ...	(GetProcs[j].ReadProc)[k];	-- The proc can be computed.
	In 19, 26. •TRASH may be written as NULL, ~ as :.		file.Read[buffer~b, count~k];	-- wFile.Read[file, b, k] (obj
27.1	applEn ::= ! enChoice[9]; ... -- In 19, 26.	BUT {enChoice; ... }	f[i~3, j~ , k~TRASH]; f[i~3, k~TRASH];	-- j and k may be trash (see d
			f[3, , TRASH];	-- Likewise, if i, j, and k ar
28	if ::= IF e_1 THEN e_2 (ELSE e_3 |)	IF e_1 THEN e_2 ELSE (e_3 | NULL)	i_(IF j<3 THEN 6 ELSE 8);	-- An IF with results must have
			IF k NOT IN Range THEN RETURN[7];	
29	select ::= SELECT e FROM	LET selector(~e IN	SELECT f[j] FROM	-- SELECT expressions are also
	choice; ... endChoice	choice ... endChoice	<7=>{...};	-- wt:INT~f [j]; IF t<7 THEN {.
	The ";" is "," in an expression; also ";" is a separator for repetitions of the choice. In 32, 34.		IN [7..8]=>{...};	-- 7, 8=> or =7, =8=>{...} is
30	choice ::= ((| relOp[22]) e_1), !.. => e	IF (selector((= | relOp) e_1) OR ...) THEN e	NOT <=8=>{...};	-- ENDCASE=>{...} is the same h
31	endChoice ::= ENDCASE (| => e_3)	ELSE (NULL | e_3)	ENDCASE=>ERROR;	-- Redundant: choices are exha
	In 29, 32, 34.			
32	safeSelect ::= WITH e SELECT FROM	LET v(~e IN	WITH r SELECT FROM	-- Assume r: REF ANY in this ex
	safeChoice; ... endChoice[31]	safeChoice ... endChoice	rInt: REF INT=>RETURN[Gcd[rInt^, 17]];	-- rInt is declared in this ch
33	safeChoice ::= n : t => e_2	IF ISTYPENOTNIL[v(, t] THEN LET n : t_NARROW[v(, t] IN e_2	rReal: REF REAL=>RETURN[Floor[Sin[rReal^]]];	
34	•withSelect ::= WITH (n_1 ~~ e_1 | • e_1)	OPEN v(~~e_1 IN LET n(~(n_1 | NIL), type(~v(,	ENDCASE=>RETURN[IF r=NIL THEN 0 ELSE 1]	-- Only the REF ANY r is known
	SELECT (| ,e_{11}) FROM	selector(~(e_1.TAG | e_{11}) IN withChoice ... endChoice	nr: REF Node[52]~...; WITH dn~~nr SELECT FROM	-- See rule 52 for the variant
	withChoice; ... endChoice[31]	-- e_{11} must be defaulted except for a COMPUTED variant	binary=>{nr_dn.b};	-- dn is a Node.binary in this
	•The ~~ may be written as :.		unary=>{nr_dn.a};	-- dn is a Node.unary in this
35	•withChoice ::= n_2 => e_2 |	IF selector(=n_2 THEN OPEN	ENDCASE=>{nr_NIL};	-- dn is just a Node here.
	n_2, n_2, !.. => e_2	(BINDP[n(, LOOPHOLE[v(,type([n_2]]] | BINDP[n(, v(]) IN e_2		
36	type ::= typeName | builtInType | typeCons		P: PROC[b: Buffer[1].Handle,	-- A type from an interface.
37	typeName ::= n_1 | typeName . n_2 |		i: INT_TEXT[20].SIZE];	-- A bound sequence; only in s
	•typeName [e] | •n_2 typeName	typeName.SPECIALIZE[e] | typeName . n_2		
	In 19, 25, 36, 40.1, 49.	--n_2 names a variant.		
38	builtInType ::= INT | REAL | TYPE | ATOM | MONITORLOCK | CONDITION |			
	µ ?,UNCOUNTED ZONE | •,MDSZone | •, LONG CARDINAL | •, ?LONG UNSPECIFIED -- See Table 4 2.			
	TYPE only as t in a b or an interface's d. INTEGER, CARDINAL, NAT, TEXT, STRING, BOOL, CHAR are predefined.			
39	typeCons ::= subrange[25] | paintedTC[40.1] | transferTC[41] | arrayTC[44] | seqTC[45] | ,descriptorTC[45.1] |		TypeIndex: TYPE~[0..256);	-- A subrange type.
	refTC[46] | listTC[47] | ,pointerTC[48] | •,relativeTC[49] | recordTC[50] | unionTC[52] | enumTC[54] | defaultTC[55]		BinaryNode: TYPE~Node[52].binary;	-- A bound variant type.
40	varTC ::= (| READONLY | VAR) t | ANY	(ANY | READONLY | VAR) t | ANY		
	In 11, 45 48. ANY only in refTC. VAR only in interface decl.			
40.1	,paintedTC ::= typeName PAINTED t	REPLACEPAINT[in: t, from: typeName]	HV: TYPE~Interface.HistValue PAINTED	-- See 13 for use.
	typeName must be an opaque type, t recordTC or enumTC.		RECORD[...]	
41	transferTC ::=?safety[4] xfer ?drType	MKXFERTYPE[drType, flavor~xfer]	Enumerate: PROC[	
41.1	xfer ::= PROCEDURE | PROC | PORT |		l: RL,	
	PROCESS | SIGNAL | ERROR | PROGRAM		p: PROC[x: REF ANY] RETURNS [stop: BOOL]]	
42	drType ::= ?$fields_1$ RETURNS $fields_2$	domain~$fields_1$, range~$fields_2$	RETURNS [stopped: BOOL];	
	No domain for PROCESS. In 3, 41.		p2:PROCESS RETURNS[i:INT]_FORK stream.Get;	
43	fields ::= [d[11], ...] | [t, ...] | ANY		failed: ERROR [reason: ROPE]~CODE;	
	ANY only in drType. In 42, 50, 52.			
44	arrayTC ::= ?µPACKED ARRAY ?t_1 OF t_2	MKARRAY[domain~t_1, range~t_2]	Vec: TYPE=ARRAY [0..maxVecLen) OF REF TEXT;	
45	seqTC ::=?µPACKED SEQUENCE tag[53] OF t	MKSEQUENCE[domain~tag, range~t]	Chars: TYPE~RECORD [text: PACKED SEQUENCE	-- A record with just a sequen
	Legal only as last type in a recordTC or unionTC.		len: [0..LAST[INTEGER]] OF CHAR]; ch: Chars;	-- ch.text[i] or ch[i] refers
45.1	,descriptorTC ::=		v: Vec~ALL[NIL];	
	?LONG DESCRIPTOR FOR varTC[40]	MKARRAYDESCR[arrayType~varTC]	dV: DESCRIPTOR FOR ARRAY OF REF TEXT~	
	varTC must be an array type.		DESCRIPTOR[v];	
46	refTC ::= REF (varTC[40] |)	MKREF[target~(varTC | ANY)]	ROText: TYPE~REF READONLY TEXT;	-- NARROW[rl.first, ROText]^ is
47	listTC ::= LIST OF varTC[40]	MKLIST[range~varTC]	RL: TYPE~LIST OF REF READONLY ANY; rl:RL;	-- READONLY TEXT (or error).
48	,pointerTC::=?LONG ?ORDERED ?•BASE	MKPOINTER[target~varTC] |	UnsafeHandle: TYPE~LONG POINTER TO Vec[44];	
	POINTER ?•subrange[25] ?(TO varTC[40]) |			
	•POINTER TO FRAME [n]	MKINSTANCETYPE[n]		
	Subrange only in a relativeTC; no typeName[37] on it.			
49	•,relativeTC ::= typeName[37] RELATIVE t	MKRELATIVE[range~t, baseType~typeName]		
	t must be a pointer or descriptor type, typeName a base pointer type.			
50	recordTC ::=?access[12] (		Cell: TYPE~RECORD[next: REF Cell, val: ATOM];	
	?MONITORED RECORD fields[43] |	MKRECORD[fields] |	Status: TYPE~MACHINE DEPENDENT RECORD [	-- Don't omit the field positi
	„ MACHINE DEPENDENT RECORD		channel (0: 8..10): [0..nChannels),	-- nChannels < 8.
	(mdFields | •fields[43]))	MKMDRECORD[mdFields | fields]	device (0: 0..3): DeviceNumber,	-- DeviceNumber held in < 4 bi
51	„mdFields ::= [((n pos), ... :	MKMDFIELDS[LIST[(LIST[([$n, pos]), ...] , t	stopCode (0: 11..15): Color, fill (0: 4..7): BOOL,	-- No gaps allowed, but any or
	?•access[12] t), ...]		command (1: 0..31): ChannelCommand];	-- Bit numbers >16 OK; fields
51.1	„pos ::= (e_1 ?(: e_2 .. e_3)) In 51, 53.	MKPOSITION[firstWord~e_1, firstBit~e_2, lastBit~e_3]		-- word boundaries only if w
52	unionTC ::= SELECT tag FROM	MKUNION[selector~tag, variants~LIST[	Node: TYPE~MACHINE DEPENDENT RECORD [	-- rands is a union or variant
	(n, ...=>(fields[43] | mdFields[51] | •NULL)), ...	([labels~LIST[$n, ...], value~fields])	type (0: 0..15): TypeIndex, rator (1: 0..13): Op,	-- This is the common part.
	?, ENDCASE		rands (1: 14..79): SELECT n (1: 14..15): * FROM	-- Both union and tag have pos
	Legal only as last type in a recordTC or unionTC.		nonary=>[],	-- Type of n is {nonary, unary
53	tag ::= (n („pos[51.1] |) : ?•access[12] (| [$n, (pos | NIL)] | $COMPUTED(| $OVERLAID( )		unary=>[a (1: 16..47): REF Node],	-- Can use same name in severa
	µ,COMPUTED | µ,OVERLAID) (t | *)	(t | TYPEFROMLABELS)]	binary=>[a (1:16..47), b(1:48..79)]: REF Node]	-- At least one variant must f
	In 44, 52. * only in unionTC[52].		ENDCASE];	
54	enumTC ::= { n, ... } |	MKENUMERATION[LIST[$n, ...]] |	Op: TYPE~{plus, minus, times, divide };	
	MACHINE DEPENDENT {((n |) (| n), ... }	MKMDENUMERATION[LIST[([($n \| NIL), e] \| [$n, 1]), ...]]	Color: TYPE~MACHINE DEPENDENT {	-- A Color value takes 4 bits;
			red(0), green, blue(4), (15)}; c: Color;	
55	defaultTC ::=	CHANGEDEFAULT[type~t, (	-- Except as noted, a constructor or application must mention each	
	t _ |	proc~NIL, trashOK~FALSE] |	Q: TYPE~RECORD[	-- Otherwise there's a compile
	t _ e |	proc~INLINE l IN e, trashOK~FALSE] |	i: INT,	-- Q[], Q[i~] trash i (not in
	µt _ e TRASH |	proc~INLINE l IN e, trashOK~TRUE] |	j: INT_,	-- No defaulting or trash for
	µt _ TRASH	proc~t.Trash, trashOK~TRUE])	k: INT_3,	-- Q[], Q[k~] leave k=3.
	defaultTC legal only as the type in a decl in a body[9] or field[43] (n: t _ e), in a TYPE binding[13] in NEW. Note the terminal |.		l: INT_3 | TRASH,	-- As k, but Q[l~TRASH] trashes
	•TRASH may be written as NULL.		m: INT_TRASH];	-- Q[], Q[m~] trash m.
56	name ::= letter (letter | digit)...	-- But not one of the reserved words in Table 3.	x, x1, x59y, longNameWithSeveralWords: INT;	
57	literal ::= num (| D | B) ?num) |	-- INT literal, decimal if radix omitted or D, octal if B	n: INT~1+12D+2B3+2000B	-- = 1+12+1024+1024
	digit (digit | A | B | C | D | E | F) ... (H ?num |	-- INT literal in hex; must start with digit. |	+1H+0FFH;	-- +1+255
	?num . num ?exponent |	-- REAL as a scaled decimal fraction; note no trailing dot.	r: REAL~0.1+.1+1.0E 1	-- = 0.1+0.1+0.1
	num exponent |	-- With an exponent, the decimal point may be omitted. |	+1E 1;	-- +0.1
	' extendedChar | • digit !.. C	-- CHAR literal; the C form specifies the code in octal.	a1: ARRAY [0..3] OF CHAR~['x, '\N, '\', '\141];	
	" extendedChar ... " ?•L	[('extendedChar), ...] -- Rope.ROPE, TEXT, or STRING.	r2: ROPE~"Hello.\N...\NGoodbye\F";	
	$ n	-- ATOM literal.	a2: ATOM~$NameInAnAtomLiteral;	
58	exponent ::= (e | E) ?(+ |) num	-- Optionally signed decimal exponent.		
59	num ::= digit !..			
60	extendedChar ::= space | \ extension | anyCharNot'"Or\			
61	extension ::= $digit_1$ $digit_2$ $digit_3$ |	-- The character with code $digit_1$ $digit_2$ $digit_3$ B. |		
	(N | n) | (R | r) | (T | t) | (B | b) |	-- CR, '\015 | TAB, '\011 | BACKSPACE, '\010 |		
	(F | f) | (L | l) | ' | " | \	-- FORMFEED, '\014 | LINEFEED, '\012 | ' | " | \		

Notation: item \| item=choose one; ?item=zero or one item; terminals: SMALL CAPS, punctuation other than bold ()?\| (parens are terminals only in rules 19,
item s ...=zero or more items, separated by s; item s !..=one or more items, separated by s; with s=";", a trailing ";" is opti

Abbreviated non-terminals: b=binding[13]; d=declaration[11]; e=expression[19]; n=name[56]; s=statement[14]; t=type[36].

Comments: •=obsolete; µ=efficiency hack; ,=unsafe; „=machine dependent; n[56] means n is defined in rule 56; n_2=n (the subscript is only for the desu