11.11.1. *Bare and decorated interval literals.*

This subclause defines an **interval literal**: a (text) string that denotes a bare or decorated interval. This entails defining a **number literal**: a string, occurring within an interval literal, that denotes an extended-real number.

The interval denoted by an interval literal is its *value* (as an interval literal). Any other string has no value in that sense. For convenience a string is called *valid* or *invalid* (as an interval literal) according as it does or does not have such a value. The usage of value, valid and invalid for number literals is similar.

[*Note. The notion of number literal is only used in this subclause. Interval literals are used as input to* text2interval, *see* §11.11.8. *The definition of interval literal is not intended to constrain the syntax and semantics that a language might use to denote intervals in other contexts.*]

This definition of an interval literal s is placed in Level 2 because its use is not mandatory at Level 1, see §9.6.1. However the value of s is a Level 1 interval x. Within s, conversion of a number literal to its value shall be done as if in infinite precision, ignoring any default or explicit language-defined precision specification. Conversion of x to a Level 2 datum y is a separate operation. In all cases y shall contain x; typically y is the $\mathbb{T}$-hull of x for some interval type $\mathbb{T}$.

[*Examples.*

– *The interval denoted by the literal* [1.2345] *is the Level 1 single-point interval* $x = [1.2345, 1.2345]$. *However the result of* $\mathbb{T}$-text2interval("[1.2345]"), *where* $\mathbb{T}$ *is the 754 infsup* binary64 *type, is the interval, approximately* $[1.2344999999999999, 1.2345000000000002]$, *whose endpoints are the nearest* binary64 *numbers either side of* 1.2345.

– *Suppose the host language is C/C++. There, conversion of a number literal to a floating point format is considered part of its value, and a "floating-suffix" can be added to change this. E.g.,* 1.2345f *denotes 1.2345 rounded to the nearest* float *(single-precision)—instead of to the nearest* double, *which is the default. This is ignored within an interval literal:* 1.2345 *and* 1.2345f *both denote the mathematical number 1.2345.*

]

The following forms of number literal shall be supported:

(1) A number literal[3] in the syntax provided by the host language of the implementation, in decimal or any other supported radix.

(2) Either of the strings inf or infinity, ignoring case, optionally preceded by +, with value $+\infty$; or preceded by -, with value $-\infty$.

(3) A string in the hexadecimal-significand form of IEEE 754-2008, §5.12.3, with the value specified there.

(4) A string p/q, that is p and q separated by the / character, where p, q are decimal integer strings, with p optionally signed and $q > 0$. Its value is the exact rational number p/q.

[*Note. These categories need not be mutually exclusive, e.g., in C/C++, form (3) is included in (1).*]

Bare intervals. The forms of bare interval literal that shall be supported are listed below. To simplify stating the needed constraints, e.g. $l \le u$, the number literals l, u, m, r are identified with their values. Space shown between elements of a literal, below, denotes zero or more characters that count as whitespace in the host language. [*Note. Some characters, e.g., newline, might be whitespace in some contexts but not others.*]

Constants: The string [empty], ignoring case, whose value is the empty set $\emptyset$; and the string [entire], ignoring case, whose value is the whole line $\mathbb{R}$.

Inf-sup form: A string [l , u] where l and u are number literals with $l \le u$, $l < +\infty$ and $u > -\infty$, see §9.2. Its value is the mathematical interval $[l, u]$. A string [x] is equivalent to [x , x].

Uncertain form: a string $m?rue$ where: m is a decimal number literal of form (1) above, without exponent; r is empty or is a non-negative decimal integer *ulp-count*; u is empty or is a *direction character*, either u (up) or d (down); and e is empty or is a decimal *exponent field* for the number m. No whitespace is permitted within the string.

One ulp equals one unit in the last place of the number m as written. The literal $m?$ by itself denotes m with a symmetrical uncertainty of $\frac{1}{2}$ulp, that is the interval

---

[3]"floating constant" in C, "real literal constant" in Fortran, "real numeric literal" in Ada, etc.

$[m - \frac{1}{2}\texttt{ulp}, m + \frac{1}{2}\texttt{ulp}]$. The literal $m\texttt{?}r$ denotes m with a symmetrical uncertainty of r ulps, that is $[m - r \times \texttt{ulp}, m + r \times \texttt{ulp}]$. Adding $\texttt{d}$ (down) or $\texttt{u}$ (up) converts this to uncertainty in one direction only, e.g. $m\texttt{?d}$ denotes $[m - \frac{1}{2}\texttt{ulp}, m]$ and $m\texttt{?}r\texttt{u}$ denotes $[m, m + r \times \texttt{ulp}]$. Finally the exponent field if present multiplies the whole interval by the appropriate power of 10, e.g. $m\texttt{?}r\texttt{u}e$ denotes $10^e \times [m, m + r \times \texttt{ulp}]$.

[*Examples. Assuming the common types of decimal number literals are provided:*

Form	Literal	Value
Inf-sup	`[1.e-3, 1.1e-3]`	$[0.001, 0.0011]$
	`[-Inf, 2/3]`	$[-\infty, 2/3]$
Uncertain	`3.56?`	$[3.555, 3.565]$
	`3.56?1`	$[3.55, 3.57]$
	`3.560?2`	$[3.558, 3.562]$
	`3.560?2u`	$[3.560, 3.562]$
	`3.56?1e+1`	$[35.5, 35.7]$

]

A formal description of bare interval literals is by the following grammar (using the notation of 754§5.12.3) which defines a bareIntvlLiteral, subject to the constraints on l, u, m, r stated above. anycase(s) matches all upper/lower case variants of the string s, e.g. anycase("ab") matches any of "ab", "Ab", "aB", "AB".

⚠ Meta-characters are temporarily colored red, to help spot any meta-syntax-errors in this grammar.

sign	`[+-]`
digit	`[0123456789]`
natural	`{digit} +`
langNumLit	`{number literal of host language}`
langNumLitMant	`{mantissa of number literal of host language}`
langNumLitExp	`{exponent of number literal of host language}`
sp	`{whitespace character of host language} *`
infLit	`{sign} ? (anycase("inf")	anycase("infinity"))`
hexNumLit	`{see 754§5.12.3}`
ratNumLit	`{sign} ? {digit} + "/" {digit} +`
numberLiteral	`( {langNumLit} | {infLit} | {hexNumLit} | {ratNumLit} | )`
pointIntvl	`"[" {sp} {numberLiteral} {sp} "]"`
infSupIntvl	`"[" {sp} {numberLiteral} {sp} "," {sp} {numberLiteral} {sp} "]"`
uncertIntvl	`{langNumLitMant} "?" {natural} ? [du]? {langNumLitExp} ?`
constIntvl	`(anycase("[empty]")	anycase("[entire]"))`
bareIntvlLiteral	`( {pointIntvl} | {infSupIntvl} | {uncertIntvl} | {constIntvl} )`

Decorated intervals. The syntax of decorated interval literals is given by extending the above grammar as follows, to define a decoratedIntvlLiteral.

connectChar	`{language-defined character}`
decorationLit	`( "ill" | "trv" | "def" | "dac" | "com" )`
decoratedIntvlLiteral	`{bareIntvlLiteral} {connectChar} {decorationLit}`

That is, s is a bare interval literal sx, followed by a language-defined single character c followed by a 3-character decoration string sd. It is recommended that c be the underscore '_', which is assumed here. The string sd is one of $\texttt{ill}$, $\texttt{emp}$, $\texttt{trv}$, $\texttt{def}$, $\texttt{dac}$ or $\texttt{com}$ representing the corresponding decoration dx. If sx has the value x, and if x_{dx} is a permitted combination according to §10.4, then s has the value x_{dx}. Otherwise s has no value as a decorated interval literal.