12.11. Interval and number literals.

12.11.1. *Overview.*

This subclause defines an **interval literal**: a (text) string that denotes a bare or decorated interval. This entails defining a **number literal**: a string within an interval literal that denotes an extended-real number; if it denotes a finite integer it is an **integer literal**.

The bare or decorated 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 and integer literals is similar.

Interval literals are used as input to `textToInterval` §12.12.8 and in the Input/Output clause §13. In this standard, number (and integer) literals are only used within interval literals. The definitions of literals are not intended to constrain the syntax and semantics that a language might use to denote numbers and intervals in other contexts.

The definition of an interval literal $\boldsymbol{s}$ is placed in Level 2 because its use is not mandatory at Level 1, see §10.6.1. However the value of $\boldsymbol{s}$ is a bare or decorated Level 1 interval $\boldsymbol{x}$. Within $\boldsymbol{s}$, conversion of a number literal to its value shall be done as if in infinite precision. Conversion of $\boldsymbol{x}$ to a Level 2 datum $\boldsymbol{y}$ is a separate operation. In all cases $\boldsymbol{y}$ shall contain $\boldsymbol{x}$; typically $\boldsymbol{y}$ is the $\mathbb{T}$-hull of $\boldsymbol{x}$ for some interval type $\mathbb{T}$.

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

The case of alphabetic characters in interval and number literals, including decorations, is ignored. It is assumed here that they have been converted to lowercase. (E.g., `inf` is equivalent to `Inf` and `[1,2]_dac` is equivalent to `[1,2]_DAC`.)

12.11.2. *Number literals.*

An integer literal comprises an optional sign and (i.e., followed by) a nonempty sequence of decimal digits.

The following forms of number literal shall be supported.

(a) A decimal number. This comprises an optional sign, a nonempty sequence of decimal digits optionally containing a point, and an optional exponent field comprising `e` and an integer literal. The value of a decimal number is the value of the sequence of decimal digits with optional point multiplied by ten raised to the power of the value of the integer literal, negated if there is a leading `-` sign.

(b) A number in the hexadecimal-floating-constant form of the C99 standard (ISO/IEC9899, N1256, §6.4.4.2), equivalently hexadecimal-significand form of IEEE 754-2008, §5.12.3. This comprises an optional sign, the string `0x`, a nonempty sequence of hexadecimal digits optionally containing a point, and an optional exponent field comprising `p` and an integer literal. The value of a hexadecimal number is the value of the sequence of hexadecimal digits with optional point multiplied by two raised to the power of the value of the integer literal, negated if there is a leading `-` sign.

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

(d) A rational literal p `/` q, that is p and q separated by the `/` character, where p, q are decimal integer literals, with q positive. Its value is the exact rational number p/q.

An implementation may support a more general form of integer and/or number literal, e.g., in the syntax of the host language of the implementation. It may restrict the support of literals, by relaxing conversion accuracy of hard cases: rational literals, long strings, etc., converting such literals to Entire, for example. It shall document such restrictions.

By default the syntax shall be that of the default locale (C locale); locale-specific variants may be provided.

12.11.3. *Unit in last place.* The "uncertain form" of interval literal, below, uses the notion of the *unit in the last place* of a number literal $\boldsymbol{s}$ of some radix b, possibly containing a point but without an exponent field. Ignoring the sign and any radix-specifying code (such as `0x` for hexadecimal), $\boldsymbol{s}$ is a nonempty sequence of radix-b digits optionally containing a point. Its *last*

place is the integer $p = -d$ where $d = 0$ if $\boldsymbol{s}$ contains no point, otherwise d is the number of digits after the point. Then $\texttt{ulp}(\boldsymbol{s})$ is defined to equal b^p. When context makes clear, "x ulps of $\boldsymbol{s}$" or just "x ulps", is used to mean $x \times \texttt{ulp}(\boldsymbol{s})$. [*Example. For the decimal strings* `123` *and* `123.` *, as well as* `0` *and* `0.` *, the last place is* 0 *and one ulp is* 1*. For* `.123` *and* `0.123` *, as well as* `.000` *and* `0.000` *, the last place is* −3 *and one ulp is* 0.001.]

12.11.4. *Bare intervals.*

The following forms of bare interval literal shall be supported. 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 denotes zero or more space characters.

(a) Special values: The strings `[ ]` and `[ empty ]`, whose bare value is Empty; the string `[ entire ]`, whose bare value is Entire.

(b) Inf-sup form: A string `[` l `,` u `]` where l and u are number literals with $l \le u$, $l < +\infty$ and $u > -\infty$, see §10.2 and the note on difficulties of implementation §12.12.8. Its bare value is the mathematical interval $[l, u]$. A string `[` x `]` is equivalent to `[` x `,` x `]`.

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

With `ulp` meaning $\texttt{ulp}(m)$, the literal m`?` by itself denotes m with a symmetrical uncertainty of half an ulp, that is the interval $[m - \frac{1}{2}\texttt{ulp}, m + \frac{1}{2}\texttt{ulp}]$. The literal m`?`r denotes m with a symmetrical uncertainty of r ulps, that is $[m - r \times \texttt{ulp}, m + r \times \texttt{ulp}]$. Adding `d` (down) or `u` (up) converts this to uncertainty in one direction only, e.g. m`?d` denotes $[m - \frac{1}{2}\texttt{ulp}, m]$ and m`?r u` denotes $[m, m + r \times \texttt{ulp}]$. Uncertain form with radius empty or *ulp-count* is adequate for narrow (and hence bounded) intervals, but is severely restricted otherwise. Uncertain form with radius `?` is for unbounded intervals, e.g. m`??d` denotes $[-\infty, m]$, m`??u` denotes $[m, +\infty]$ and m`??` denotes Entire with m being like a comment. The exponent field if present multiplies the whole interval by 10^e, e.g. m `?`r`u e`e denotes $10^e \times [m, m + r \times \texttt{ulp}]$.

12.11.5. *Decorated intervals.*

Decorated intervals literals may denote either bare or decorated interval value depending on context. The following forms of decorated interval literal shall be supported.

(a) The string `[ nai ]`, with the bare value Empty and the decorated value $\text{Empty}_{\texttt{ill}}$.

(b) A bare interval literal $\boldsymbol{sx}$.

If $\boldsymbol{sx}$ has the bare value $\boldsymbol{x}$, then $\boldsymbol{sx}$ has the decorated value $\texttt{newDec}(\boldsymbol{x})$ §11.5. Otherwise $\boldsymbol{sx}$ has no decorated value.

(c) A bare interval literal $\boldsymbol{sx}$, an underscore "`_`", and a 3-character decoration string $\boldsymbol{sd}$; where $\boldsymbol{sd}$ is one of `trv`, `def`, `dac` or `com`, denoting the corresponding decoration dx.

If $\boldsymbol{sx}$ has the bare value $\boldsymbol{x}$, and if $\boldsymbol{x}_{dx}$ is a permitted combination according to §11.4, then $\boldsymbol{s}$ has the bare value $\boldsymbol{x}$ and the decorated value $\boldsymbol{x}_{dx}$. Otherwise $\boldsymbol{s}$ has no value as a decorated interval literal.

[*Examples. Table 12.2 illustrates valid portable interval literals. These strings are not valid portable interval literals:* `empty`, `[5?1]`, `[1_000_000]`, `[ganz]`, `[entire!comment]`, `[inf]`, `5???u`, `[nai]_ill`, `[]_ill`, `[]_def`, `[0,inf]_com`.]

12.11.6. *Grammar for portable literals.*

Portable literals permit exchange between different implementations.

The syntax of portable integer and number literals, and of portable bare and decorated interval literals, is defined by integerLiteral, numberLiteral, bareIntvlLiteral and intervalLiteral respectively, in the grammar in Table 12.3, which uses the notation of 754§5.12.3. Lowercase is assumed, i.e., a valid string is one that after conversion to lowercase is accepted by this grammar. `\t` denotes the TAB character.

The constructor `textToInterval` §12.12.8, 13.2 of any implementation shall accept any portable interval. Implementation may restrict support of some input strings (too long strings or strings with a rational number literal). Nevertheless, the constructor shall always return a Level 2 interval (possibly Entire in this case) that contains the Level 1 interval.

Form	*Literal*	*Exact decorated value*
Special	`[ ]`	$\mathsf{Empty}_{\mathtt{trv}}$
	`[entire]`	$[-\infty, +\infty]_{\mathtt{dac}}$
Inf-sup	`[1.e-3, 1.1e-3]`	$[0.001, 0.0011]_{\mathtt{com}}$
	`[-Inf, 2/3]`	$[-\infty, 2/3]_{\mathtt{dac}}$
Uncertain	`3.56?1`	$[3.55, 3.57]_{\mathtt{com}}$
	`3.56?1e2`	$[355, 357]_{\mathtt{com}}$
	`3.560?2`	$[3.558, 3.562]_{\mathtt{com}}$
	`3.56?`	$[3.555, 3.565]_{\mathtt{com}}$
	`3.560?2u`	$[3.560, 3.562]_{\mathtt{com}}$
	`-10?`	$[-10.5, -9.5]_{\mathtt{com}}$
	`-10?u`	$[-10.0, -9.5]_{\mathtt{com}}$
	`-10?12`	$[-22.0, 2.0]_{\mathtt{com}}$
	`-10??u`	$[-10.0, +\infty]_{\mathtt{dac}}$
	`-10??`	$[-\infty, +\infty]_{\mathtt{dac}}$
NaI	`[nai]`	$\mathsf{Empty}_{\mathtt{ill}}$
Decorated	`3.56?1_def`	$[3.55, 3.57]_{\mathtt{def}}$

TABLE 12.2. Portable interval literal examples.

An implementation may support interval literals of more general syntax (for example, with underscores in significand). In this case there shall be a value of conversion specifier *cs* that restricts output strings of `intervalToText` §13.3 to the portable syntax.

decDigit	`[0123456789]`
nonzeroDecDigit	`[123456789]`
hexDigit	`[0123456789abcdef]`
spaceChar	`[ \t]`
natural	{decDigit}+
sign	`[+-]`
integerLiteral	{sign}? {natural}
decSignificand	{decDigit}* "." {decDigit}+ | {decDigit}+ "." | {decDigit}+
hexSignificand	{hexDigit}* "." {hexDigit}+ | {hexDigit}+ "." | {hexDigit}+
decNumLit	{sign}? {decSignificand} ("e" {integerLiteral})?
hexNumLit	{sign}? "0x" {hexSignificand} ("p" {integerLiteral})?
infNumLit	{sign}? ("inf" | "infinity")
positiveNatural	("0")* {nonzeroDecDigit} {decDigit}*
ratNumLit	{integerLiteral} "/" {positiveNatural}
numberLiteral	{decNumLit} | {hexNumLit} | {infNumLit} | {ratNumLit}
sp	{spaceChar}*
dir	"d" | "u"
pointIntvl	"[" {sp} {numberLiteral} {sp} "]"
infSupIntvl	"[" {sp} {numberLiteral} {sp} "," {sp} {numberLiteral} {sp} "]"
radius	{natural} | "?"
uncertIntvl	{sign}? {decSignificand} "?" {radius}? {dir}? ("e" {integerLiteral})?
emptyIntvl	"[" {sp} "]" | "[" {sp} "empty" {sp} "]"
entireIntvl	"[" {sp} "entire" {sp} "]"
specialIntvl	{emptyIntvl} | {entireIntvl}
bareIntvlLiteral	{pointIntvl} | {infSupIntvl} | {uncertIntvl} | {specialIntvl}
NaI	"[" {sp} "nai" {sp} "]"
decorationLit	"trv" | "def" | "dac" | "com"
intervalLiteral	{NaI} | {bareIntvlLiteral} | {bareIntvlLiteral} "_" {decorationLit}

TABLE 12.3. Grammar for literals: integer literal is integerLiteral, number literal is numberLiteral, bare interval literal is bareIntvlLiteral and decorated interval literal is intervalLiteral.