9.4. Common interval and number literals

This subclause defines a flavor-independent syntax of literals for common intervals, that may be extended by a flavor to include its non-common intervals.

9.4.1. Overview

A **bare interval literal** of a flavor is a (text) string that denotes a bare Level 1 interval of the flavor. A **decorated interval literal** of a flavor is a (text) string that denotes a decorated Level 1 interval of the flavor. Bare interval literals and decorated interval literals are collectively called **interval literals**. 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**.

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

Bare and decorated interval literals are used as input to bare and decorated versions of `textToInterval` in 9.5. 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 value of $\boldsymbol{s}$ is a bare or decorated Level 1 interval $\boldsymbol{x}$. Level 2 operations with interval literal inputs are evaluated according to 7.5.3; typically they return the $\mathbb{T}$-hull of $\boldsymbol{x}$ for some interval type $\mathbb{T}$.

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

Each flavor defines a set of **portable** bare and decorated interval literals. Portable interval literals shall be valid interval literals in each implementation of the flavor. An implementation may support a more general form of valid interval literals, e.g., using number literals in the syntax of the host language of the implementation. It may restrict the support of portable interval literals at Level 2, by relaxing conversion accuracy of hard cases: rational number literals, long strings, etc. It shall document such restrictions.

This subclause defines a set of **common** bare and decorated interval literals. Their common values are common bare or decorated intervals. They shall be portable interval literals in all flavors. The value of a common interval literal in a flavor is the same as its common value, modulo the embedding map. Portable interval literals of a flavor may contain literals other than common. Some of them denote non-common intervals of the flavor. Others may be alternative notation for common intervals.

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., `[1,1e3]_com` is equivalent to `[1,1E3]_COM`.)

9.4.2. Common number literals

An integer literal comprises an optional sign and (i.e., followed by) a nonempty sequence of decimal digits. A sign is a plus sign `+` or a minus sign `-`.

The following forms of number literal shall be provided by all flavors.

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 minus 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 Std 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 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 minus sign.

c) 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.

These are common number literals. A flavor may define portable integer and/or number literals other than common.

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

9.4.3. 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*.]*

9.4.4. Common bare intervals

The following forms of bare interval literal shall be supported in all flavors. To simplify stating the needed constraints, e.g., $l \le u$, the number literals l, u, m, r are identified with their values.

a) Inf-sup form: A string `[` l `,` u `]` where l and u are common number literals with $l \le u$. Its common bare value is the common interval $[l, u]$. A string `[` m `]` with number literal m is equivalent to `[` m `,` m `]`.

b) Uncertain form: a string m `?` r u E where: m is a decimal number literal of form a) in 9.4.2, without exponent; r is empty or is a non-negative decimal integer literal *ulp-count*; 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 $[m, m + r \times \texttt{ulp}]$. The exponent field if present multiplies the whole interval by 10^e, e.g., m`?`r`u``e`e denotes $10^e \times [m, m + r \times \texttt{ulp}]$.

These are common bare interval literals. A flavor may define portable bare interval literals other than common. Some of them denote non-common bare intervals of the flavor. Others may be alternative notation for common intervals.

[Examples. Table 9.2 illustrates common bare interval literals. These strings are not common bare interval literals: `[1_000_000]`, `[1.0 e3]`, `[1,2!comment]`, `[2,1]`, `[5?1]`, *@5 ?1@*, `5??u`, `[]`, `[empty]`, `[ganz]`, `[1,]`, `[1,inf]`.*]*

9.4.5. Portable decorated intervals

The following forms of portable decorated interval literal shall be supported in all flavors.

a) A portable bare interval literal $\boldsymbol{sx}$. If $\boldsymbol{sx}$ has the bare value $\boldsymbol{x}$, then $\boldsymbol{sx}$ has the decorated value $\texttt{newDec}(\boldsymbol{x})$ 8.1. Otherwise $\boldsymbol{sx}$ has no decorated value.

b) A portable bare interval literal $\boldsymbol{sx}$, an underscore "_", and a valid decoration string $\boldsymbol{sd}$. A flavor-defined valid decoration string $\boldsymbol{sd}$ denotes the corresponding decoration dx. The string `com` shall be a valid decoration string in all flavors.

TABLE 9.2. Common bare interval literal examples.

Form	Literal	Exact value
Inf-sup	`[1.e-3, 1.1e-3]`	$[0.001, 0.0011]$
	`[-0x1.3p-1, 2/3]`	$[-19/32, 2/3]$
Uncertain	`3.56?1`	$[3.55, 3.57]$
	`3.56?1e2`	$[355, 357]$
	`3.560?2`	$[3.558, 3.562]$
	`3.56?`	$[3.555, 3.565]$
	`3.560?2u`	$[3.560, 3.562]$
	`-10?`	$[-10.5, -9.5]$
	`-10?u`	$[-10.0, -9.5]$
	`-10?12`	$[-22.0, 2.0]$

If $\boldsymbol{sx}$ has the bare value $\boldsymbol{x}$, $\boldsymbol{sd}$ is a valid decoration string in a flavor with a value dx, and if $\boldsymbol{x}_{dx}$ is a permitted combination in the flavor, e.g., in the set-based flavor 11.4, then $\boldsymbol{sx\_sd}$ has the decorated value $\boldsymbol{x}_{dx}$ in the flavor. Otherwise $\boldsymbol{sx\_sd}$ has no value as a decorated interval literal in the flavor.

A common decorated interval literal is either $\boldsymbol{sx}$ or $\boldsymbol{sx}$`_com`, where $\boldsymbol{sx}$ is a common bare interval literal. Its common decorated value is $\boldsymbol{x}_{\mathtt{com}}$.

9.4.6. Grammar for common literals

The syntax of common integer and number literals and of common bare and decorated interval literals is defined by integerLiteral, numberLiteral, bareIntvlLiteral and decoratedIntvlLiteral, respectively, in the grammar in Table 9.3. 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.

Flavors define a set of portable literals as an extension of the set of common literals. [The value of a common bare literal in a flavor is its common value, modulo the embedding map. The interval part of the value of a common decorated interval literal in a flavor is its common value, modulo the embedding map. The decoration part of the value of a common decorated literal in a flavor is `com`.]

9.5. Constructors

An interval constructor by definition is an operation that creates a bare or decorated interval from non-interval data. Constructors of common bare intervals are defined in each flavor as follows.

- The operation `numsToInterval`(l, u) takes real values l and u. If the condition $l \le u$ holds, its value is the common bare interval $[l, u] = \{ x \in \mathbb{R} \mid l \le x \le u \}$. Otherwise, it has no common value.
- The bare operation `textToInterval`(s) takes a text string s. If s is a valid bare interval literal with a common value $\boldsymbol{x}$, see 9.4.4, its value is the common bare interval $\boldsymbol{x}$. Otherwise, it has no common value.
- The decorated operation `textToInterval`(s) takes a text string s. If s is a valid decorated interval literal with a common value $\boldsymbol{x}_{\mathtt{com}}$, see 9.4.5, its value is the common decorated interval $\boldsymbol{x}_{\mathtt{com}}$. Otherwise, it has no common value.

A flavor may extend the functionality of these constructors and/or provide other bare interval constructors, as appropriate to the flavor.

Each bare interval constructor shall have a corresponding decorated constructor that provides a decoration appropriate to the flavor. If the bare interval constructor has a bare common value $\boldsymbol{x}$, the decorated constructor has a decorated common value $\boldsymbol{x}_{\mathtt{com}}$. Otherwise, it has no common value.

The constructor `textToInterval` of any implementation shall accept any portable interval. An implementation may relax at Level 2 accuracy mode of some input strings (too long strings or strings with a rational number literal). Nevertheless, the constructor shall either return a Level 2 containing the Level 1 interval or signal an exception (7.5.3).