Document Number: P3003R0
Date: 2023-10-14
Revises: D3003R0
Reply to: Johel Ernesto Guerrero Pe~na
johelegp@gmail.com
JohelEGP/jegp.numbers
JohelEGP/proposals
Audience: SG6 Numerics

The design of a library of number concepts

Contents

1 Revision history [paper.revs]

1 The following changes have been applied to this paper.

(1.1) — *cv* `void` -> R0 (`2023y/October/14`, pre-Kona).

2 Introduction [paper.intro]

1 This paper presents the design of a library of number concepts. The design differs from P1813, but its feedback was taken into account. This paper is informational because the design is known to be incomplete and lacking in usage experience. We present this as a path forward towards having number concepts for constraining generic components.

3 History [paper.history]

1 After finding the pixel, Johel wrote generic components on numbers. But those components failed to abide to the C++ Core Guidelines'

(1.1) — I.9: If an interface is a template, document its parameters using concepts and

(1.2) — T.concepts: Concept rules.

2 So Johel armed himself with GCC's Concepts TS support, and eventually the LLVM fork with C++ standard concepts. He took the `Number` concept from Bjarne's presentation as a starting point. His main concerns then were to

(2.1) — specify the semantics and

(2.2) — to refactor it to support `std::chrono`'s `duration` and `time_point`.

3 He stumbled upon C++ [intro.defs] while looking for an answer to these concerns. The vocabulary of the first subject area of the Electropedia, Mathematics - General concepts and linear algebra, would serve as building blocks to solve these concerns.

4 From there, Johel reviewed the feedback on P1813 by the slides P2402 and its proposal P1673R12. Clause 4 and Clause 5 explain the actions taken and thoughts had based on these reviews.

5 The library then remained virtually unchanged for years. Johel had hardly advanced his own application in the meantime. That is what had spurred the inception and growth of the library.

6 Finally, on 2023-09-21, this library was proposed for merging into the mp-units library (mpusz/mp-units#492). This spurred a revitalization on the improvement of the design of the library.

4 Design [paper.design]

4.1 Overview [paper.design.overview]

1 We hope that this library encourages the development of well-specified generic components that operate on numbers. Clause 4 summarizes what is precisely defined in Clause 9. Additionally, we discuss what might not be evident in that reference documentation.

4.2 Type traits [paper.design.traits]

1 The type traits are mostly opt-in. They are:

(1.1) — The `number` opt-ins, `enable_number` and `enable_complex_number`.

(1.2) — The identities, `number_zero` and `number_one`.

(1.3) — The associated types, `number_difference_t` and `vector_scalar_t`.

2 They are based on existing practice. That is evident for the first two groups in Clause 9. Like `std::iter_reference_t` and `std::ranges::range_value_t`, the associated types are named *`kind-of-structure`*`_`*`associated-type`*`_t`.

4.3 Concepts [paper.design.concepts]

4.3.1 Utilities [paper.design.concepts.utils]

1 `number` is the least constrained number concept. `common_number_with` is used to relate similar types.

```
template<typename T>
concept number = enable_number_v<T> && std::regular<T>;

template<typename T, typename U>
concept common_number_with =
  number<T> && number<U> && std::common_with<T, U> && number<std::common_type_t<T, U>>;
```

4.3.2 Ordering [paper.design.concepts.order]

1 `ordered_number` requires a model of `std::totally_ordered`.

2 `number_line` is the concept that requires the complete set of increment and decrement. It is named after the term described in the Wikipedia article "Number line".

[*Note 1*: *`decrementable`* (C++ [range.iota]) doesn't work for number types.

(2.1) — It's for integer-like types (C++ [iterator.concept.winc]). Or we'd have to specialize `std::incrementable_traits` (C++ [incrementable.traits]) with a `difference_type` that is an integer-like type and that is different to `number_difference_t` (9.5.3.4).

(2.2) — It requires `std::regular`. Feedback at mpusz/mp-units#492 is that it over-constrains expression templates that can't be default-initialized.

— *end note*]

4.3.3 Arithmetic [paper.design.concepts.arithmetic]

1 Then follow the building blocks of the form `compound`$_{opt}$*`operation`*`_with`.

2 *`addition-with`* requires that `+` performs an addition (IEV 102-01-11). But addition is associative, i.e., $a + (b + c) = (a + b) + c$. The feedback from P2402 and P1673R12 is that

(2.1) — associativity is over-constraining, and

(2.2) — users generally accept "associative enough".

3 It occurred to us that we can work around this issue in the same way integer type division (C++ [expr.mul]) works. `3 / 2` is conceptually carried out in real numbers, resulting in 1.5. Then, the operator `/` yields 1.5 with the fractional part discarded, resulting in `1`.

4 *addition-with* is specified similarly to integer type division. That relaxes the associativity to "associative enough". *addition-with* requires that `a + b` performs the addition F in an unspecified set. Then, it requires that `+` yields F by mapping it to a value of the type of the result.

5 First, we introduce an unspecified set. The over-constraining associative addition is done in this unspecified set. Then, we introduce a mapping from the addition to the result of `+`. The mapping is left unspecified, so the result isn't over-constrained.

6 The operations required by other refinements of `number` are specified similarly.

4.3.4 Terse syntax [paper.design.concepts.terse]

1 Some concepts ending in `_for` just require a single concept ending in `_with` with inverted arguments.

```
template<typename T, typename U>
concept modulus_for = modulo_with<U, T>;
template<typename T, typename U>
concept vector_space_for = point_space_for<U, T>;
```

These enable the terse concept syntax.

[*Example 1*:

```
template<vector_space_for<Number> Number2> auto& operator+=(vector<Number2>);
auto operator%(modulus_for<Number> auto);
```

— *end example*]

4.3.5 Algebraic structures [paper.design.concepts.algebraic]

1 Finally, the building blocks are used to define some helpers. These helpers are used to define heterogeneous concepts on algebraic structures. There also are homogeneous counterparts with a default.

Table 1: Examples of models of algebraic structure concepts [tab:paper.design.concepts.algebraic]

Concept	Close model
`point_space`	`std::chrono::sys_seconds`
`f_vector_space`	`std::chrono::seconds`
`field_number`	`std::complex<double>`
`field_number_line`	`double`
`scalar_number`	`double` or `std::complex<double>`

5 Evolution [paper.evo]

5.1 Relax to generalize [paper.evo.relax]

1 The concepts are known to be over-constraining for templates like `boost::hana::int_c`. The feedback from mpusz/mp-units#492 suggests it is time for the hierarchy to be refactored. This will allow a wider variety of number templates that make interesting uses of the type system.

5.2 More operations [paper.evo.ops]

1 There are operations beyond those required by the definition of a vector space (IEV 102-03-01).

2 P2980R0 needs some of those to specify some quantities. Just like there are scalar quantities (IEV 102-02-19), there are vector and tensor quantities. An example is speed (IEV 113-01-33) (a scalar quantity), defined as the magnitude (IEV 102-03-23) of the velocity (IEV 113-01-32) (a vector quantity). Here are other operations from mpusz/mp-units#493:

```
template<typename T>
concept VectorRepresentation = /* ... */ &&
  requires(T a, T b, /* ... */) {
    /* ... */
    { dot_product(a, b) } -> Scalar;
    { cross_product(a, b) } -> Vector;
    { tensor_product(a, b) } -> Tensor;
    { norm(a) } -> Scalar;
  };
template<typename T>
concept TensorRepresentation = /* ... */ &&
  requires(T a, T b, /* ... */) {
    /* ... */
    { tensor_product(a, b) } -> Tensor;
    { inner_product(a, b) } -> Tensor;
    { scalar_product(a, b) } -> Scalar;
  };
```

3 Scalars, vectors, and tensors are vector spaces (IEV 102-03-01). The number concepts library has `scalar_number` (9.5.4), which represents a scalar (IEV 102-02-18). `vector_space` could be refined with these additional operations to support vectors and tensors.

4 But vector is an overloaded word. Its Wikipedia article says

> In mathematics and physics, vector is a term that refers colloquially to some quantities that cannot be expressed by a single number (a scalar), or to elements of some vector spaces.

`vector_space` can be renamed to its alternative term `linear_space`. Then, we could add `vector` for the colloquial "tuple of 2 or more scalars" and `tensor` as refinements.

5 The C++ standard library offers many more operations on arithmetic types, which are models of `scalar_number`. mp-units also offers a subset of those for `mp_units::quantity`, whose specializations model `vector_space`. The current design of this number concepts library doesn't consider generalizing this.

5.3 Mixed expressions [paper.evo.mixed.expr]

1 A *mixed expression* is a mathematical expression with more than one operation. P1673R12 describes, in particular in 10.8.4, why concepts generally are of little help.

2 The number concepts only require expressions with up to two operands. But an implementation of an algorithm might need to use mixed expressions. The unfeasible alternative would be to include all possible constraints an algorithm might use.

3 For example, the magnitude of a 2-dimensional Cartesian vector is $|\vec{v}| = \sqrt{x^2 + y^2}$ (IEV 102-03-23). Consider a C++ representation like this:

```
template<scalar_number Number>
struct vector2d {
  Number x;
  Number y;

  // A faster alternative when just ordering by magnitude.
  auto square_magnitude() { return x * x + y * y; }
};
```

`scalar_number` requires that the type of `Number` squared models `common_number_with<Number>`. But it doesn't require that adding squared `Number`s works. The best we could do is to add the additional constraint for the result of `Number` squared. It could be `point_space`, the weakest constraint, as required by the algorithm. Or it could be `scalar_number`, the same constraint as `Number`, as users would expect of the squared `Number`. There are similar expectations for the addition of the squared `Number`s. This doesn't scale to mixed expressions with more operations:

```
scalar_number auto square_magnitude() requires scalar_number<decltype(x * x)> {
  return x * x + y * y;
}
```

4 One of the points P1673R12 summarizes is:

(4.1) — We can and do use existing Standard language, like *GENERALIZED_SUM*, for expressing permissions that algorithms have.

For *GENERALIZED_SUM*, see C++ [numeric.ops]. We agree with P1673R12 on this point. But that doesn't preclude constraining algorithms with number concepts.

5 There is still value in I.9 and T.concepts. The number concepts in this library already don't require the result of addition to be associative (4.3.3). It's also possible to require that a mixed expression works as one would naturally expect. This can be done by adding an extra semantic requirement on refinements of `number`.

6 How to deal with this lies in the use of `common_number_with` in refinements of `number`. `common_number_with<T, U>` subsumes `std::common_with<T, U>` (C++ [concept.common]). Although it's not part of `std::common_with`, `std::common_type_t<T, U>` also generally behaves like `T` and `U`.

7 This is the idea for the semantic requirement on refinements of `number`. If `common_number_with<T, U>` is required, the user can use `T` in place of `U`, and the semantic constraints on `U` also apply to `T`.

8 With this requirement in mind, let's reconsider this example:

```
auto square_magnitude() { return x * x + y * y; }
```

`scalar_number<Number>` requires:

```
{ c * u } -> common_number_with<V>;
```

That's essentially:

```
{ x * x } -> common_number_with<Number>;
```

Even if the type of `Number` squared isn't the same as `Number`, the semantic constraints of `scalar_number<decltype(x * x)>` apply when the user writes one of its required expressions. In this case, the algorithm is required to work as expected.

9 It is the intent that this doesn't preclude the `+` on squared `Number`s from being ill-formed. It is also the intent that the `+` on squared `Number`s is still not required to be a total function (C++ [structure.requirements]) or to be valid for all input values (C++ [concepts.equality]). I.e., the result can still be incorrect or exit via an exception.

10 The worth of this formulation is that you can meet I.9 and T.concepts. Even if an algorithm still needs to be more specific, as with *GENERALIZED_SUM*.

6 Similar works [paper.cmp]

6.1 P1813 [paper.p1813]

1 The algorithms from `<numeric>` still don't have constrained counterparts in C++23's `std::ranges`. P1813 aimed to remedy that. Its approach to specifying algebraic structures was bottom-up. The feedback P2402 and P1673R12 gave was that this is over-constraining.

2 This paper's approach is instead top-down. Having started from Bjarne's `Number` (Clause 3), we have refactored the existing concepts as needs arise.

3 `<numeric>` algorithms work with numeric expressions or have overloads with function objects. The precise suitability of our number concepts for those constrained overloads hasn't been studied. The unconstrained version of `std::accumulate` already allows accumulating on a `std::string`. We would expect its constrained version to allow the same.

7 Acknowledgements [paper.ack]

1 I'd like to thank Mateusz Pusz for encouraging me to write a paper about this number concepts library.

2 I'd also like to thank him and the other reviewers of mpusz/mp-units#492 for their stimulating feedback. It has brought back memories of the mostly unwritten history, design and evolution of this library.

3 All the advertised improvements over previous works is thanks to the existence of P1813 and its feedback by P2402 and P1673R12. So thank you to everyone involved on those for making this possible.

8 References [paper.refs]

1 In addition to 9.2, the following documents are referred by this paper.

(1.1) — Show off the power of the new variadic dimensions system – Issue #124 – nholthaus/units

(1.2) — C++ Core Guidelines – I.9: If an interface is a template, document its parameters using concepts

(1.3) — C++ Core Guidelines – T.concepts: Concept rules

(1.4) — CppCon 2018: Bjarne Stroustrup "Concepts: The Future of Generic Programming (the future is here)"

(1.5) — IEC 60050 - International Electrotechnical Vocabulary - Welcome

(1.6) — P1673R12 A free function linear algebra interface based on the BLAS

(1.7) — mpusz/mp-units#492 feat!: add number concepts by JohelEGP

(1.8) — C++ Core Guidelines – T.21: Require a complete set of operations for a concept

(1.9) — Number line - Wikipedia

(1.10) — Boost.Hana: boost::hana::integral_constant< T, v > Struct Template Reference

(1.11) — mpusz/mp-units#493 Vector and Tensor quantities by mpusz

(1.12) — Vector (mathematics and physics) - Wikipedia

(1.13) — mp-units/src/utility/include/mp-units/math.h at v2.0.0 – mpusz/mp-units

(1.14) — Expression (mathematics) - Wikipedia

(1.15) — JohelEGP/jegp.numbers

9 API [paper.api]

1 This is a reference documentation. Any resemblance to wording is purely incidental. The interfaces are as proposed at mpusz/mp-units#492 at the time of this writing. Going forward, evolution will happen at JohelEGP/jegp.numbers.

9.1 Scope [scope]

1 This document describes the contents of the *mp-units library*.

9.2 References [refs]

1 The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

(1.1) — The C++ Standards Committee. N4892: *Working Draft, Standard for Programming Language C++*. Edited by Thomas Köppe. Available from: `https://wg21.link/N4892`

(1.2) — The C++ Standards Committee. P1841R1: *Wording for Individually Specializable Numeric Traits*. Edited by Walter E. Brown. Available from: `https://wg21.link/P1841R1`

(1.3) — The C++ Standards Committee. SD-8: *Standard Library Compatibility*. Edited by Bryce Lelbach. Available from: `https://wg21.link/SD8`

(1.4) — ISO 80000-2:2019, *Quantities and units — Part 2: Mathematics*

(1.5) — IEC 60050 (all parts), *International Electrotechnical Vocabulary (IEV)*

(1.6) — IEC 60050-102:2007/AMD3:2021, *Amendment 3 — International Electrotechnical Vocabulary (IEV) — Part 102: Mathematics — General concepts and linear algebra*

(1.7) — IEC 60050-112:2010/AMD2:2020, *Amendment 2 — International Electrotechnical Vocabulary (IEV) — Part 112: Quantities and units*

2 N4892 is hereinafter called *C++*.

3 IEC 60050 is hereinafter called *IEV*.

9.3 Terms and definitions [defs]

1 For the purposes of this document, the terms and definitions given in C++, IEC 60050-102:2007/AMD3:2021, IEC 60050-112:2010/AMD2:2020, the terms, definitions, and symbols given in ISO 80000-2:2019, and the following apply.

2 ISO and IEC maintain terminology databases for use in standardization at the following addresses:

(2.1) — ISO Online browsing platform: available at `https://www.iso.org/obp`

(2.2) — IEC Electropedia: available at `http://www.electropedia.org`

3 Terms that are used only in a small portion of this document are defined where they are used and italicized where they are defined.

9.3.1 [def.mod]
modulo
operation performed on a set for which a division (IEV 102-01-21) and an addition (IEV 102-01-11) are defined, the result of which, for elements a and b of the set, is the unique element r, if it exists in the set, such that $a = \lfloor a/b \rfloor b + r$

9.4 Specification [spec]

9.4.1 External [spec.ext]

1 The specification of the mp-units library subsumes C++ [description], C++ [requirements], C++ [concepts.equality], and SD-8, all assumingly amended for the context of this library.

[*Note 1*: This means that, non exhaustively,

(1.1) — `::mp_units2` is a reserved namespace, and

(1.2) — `std::vector<mp_units::type>` is a program-defined specialization and a library-defined specialization from the point of view of C++ and this library, respectively.

— *end note*]

2 The mp-units library is not part of the C++ implementation.

9.4.2 Categories [spec.cats]

1 Detailed specifications for each of the components in the library are in 9.5–9.5, as shown in Table 2.

2 The numbers library (9.5) describes components for dealing with numbers.

Table 2: Library categories [tab:lib.cats]

Clause	Category
9.5	Numbers library

9.4.3 Headers [headers]

1 The mp-units library provides the *mp-units library headers*, shown in Table 3.

Table 3: mp-units library headers [tab:headers.mp.units]

`<mp-units/numbers.h>`

9.4.4 Library-wide requirements [spec.reqs]

9.4.4.1 Reserved names [spec.res.names]

1 The mp-units library reserves macro names that start with `MP_UNITS`*digit-sequence*$_{opt}$`_`.

9.5 Numbers library [nums]

9.5.1 Summary [nums.summary]

1 This Clause describes components for dealing with numbers, as summarized in Table 4.

Table 4: Numbers library summary [tab:nums.summary]

	Subclause	Header
9.5.3	Traits	`<mp-units/numbers.h>`
9.5.4	Concepts	

9.5.2 Header `<mp-units/numbers.h>` synopsis [nums.syn]

```
namespace mp_units {

// 9.5.3, traits

// 9.5.3.2, number opt-ins
template<typename T>
struct enable_number;
template<typename T>
struct enable_complex_number;

template<typename T>
constexpr bool enable_number_v = enable_number<T>::value;
template<typename T>
constexpr bool enable_complex_number_v = enable_complex_number<T>::value;

// 9.5.3.3, identities
template<typename T>
struct number_zero;
template<typename T>
struct number_one;

template<typename T>
constexpr T number_zero_v = number_zero<T>::value;
template<typename T>
constexpr T number_one_v = number_one<T>::value;

// 9.5.3.4, associated types
template<typename T>
struct vector_scalar;

template<typename T>
using number_difference_t = decltype(std::declval<const T>() - std::declval<const T>());
template<typename T>
using vector_scalar_t = vector_scalar<T>::type;

// 9.5.4, concepts
template<typename T>
concept number = see below;
template<typename T, typename U>
concept common_number_with = see below;
template<typename T>
concept ordered_number = see below;
template<typename T>
concept number_line = see below;
template<typename T, typename U>
concept modulo_with = see below;
template<typename T, typename U>
concept compound_modulo_with = see below;
template<typename T, typename U>
concept modulus_for = see below;
```

```
template<typename T, typename U>
concept compound_modulus_for = see below;
template<typename T>
concept negative = see below;
template<typename T>
concept set_with_inverse = see below;
template<typename T, typename U>
concept point_space_for = see below;
template<typename T, typename U>
concept compound_point_space_for = see below;
template<typename T>
concept point_space = see below;
template<typename T, typename U>
concept vector_space_for = see below;
template<typename T, typename U>
concept compound_vector_space_for = see below;
template<typename T, typename U>
concept scalar_for = see below;
template<typename T, typename U>
concept field_for = see below;
template<typename T, typename U>
concept compound_scalar_for = see below;
template<typename T, typename U>
concept compound_field_for = see below;
template<typename T>
concept vector_space = see below;
template<typename T>
concept f_vector_space = see below;
template<typename T>
concept field_number = see below;
template<typename T>
concept field_number_line = see below;
template<typename T>
concept scalar_number = see below;

}  // namespace mp_units
```

9.5.3 Traits [num.traits]

9.5.3.1 Requirements [num.traits.reqs]

1 Subclause 9.5.3 subsumes C++ [meta.rqmts], assumingly amended for its context. Pursuant to the subsumed C++ [namespace.std] (9.4.1), each class template specified in 9.5.3 may be specialized for any numeric type (C++ [numeric.requirements]).

2 Each template `number-trait` specified in 9.5.3 has a partial specialization of the form

```
template<class T>
struct number-trait<const T> : number-trait<T> { };
```

3 Each template `number-trait` specified in 9.5.3.2 is a *Cpp17UnaryTypeTrait* with a base characteristic of `std::bool_constant<B>`. *B* is a value consistent with `number-trait`'s specification.

4 Each template specified in 9.5.3.3 is a numeric distinguished value trait (P1841R3 [num.traits.val]), except that there are no declared specializations for arithmetic or volatile-qualified types.

5 A specialization of any template `number-trait` specified in 9.5.3.4 has no members other than a `type` member that names a type consistent with `number-trait`'s specification.

9.5.3.2 number opt-ins [num.opt.ins]

```
template<typename T>
struct enable_number : see below {};
```

1 *Value*: `true` if `T` represents a number, and `false` otherwise.

2 *Default value*: `true` if `vector_scalar_t<T>` is valid, and `false` otherwise.

```
template<class... T>
struct enable_number<std::chrono::time_point<T...>> : std::true_type {};
template<>
struct enable_number<std::chrono::day> : std::true_type {};
template<>
struct enable_number<std::chrono::month> : std::true_type {};
template<>
struct enable_number<std::chrono::year> : std::true_type {};
template<>
struct enable_number<std::chrono::weekday> : std::true_type {};
template<>
struct enable_number<std::chrono::year_month> : std::true_type {};
```

```
template<typename T>
struct enable_complex_number : std::false_type {};
```

3 *Value*: `true` if `T` represents a complex number (IEV 102-02-09), and `false` otherwise.

```
template<class T>
struct enable_complex_number<std::complex<T>> : std::true_type {};
```

9.5.3.3 Identities [num.ids]

```
template<typename T>
concept inferable-identities =
  common_number_with<T, number_difference_t<T>> && std::constructible_from<T, int>;
```

```
template<typename T>
struct number_zero { see below };
```

1 *Value*: `T`'s neutral element for addition (IEV 102-01-12), if any.

2 *Default value*: If `T` models *`inferable-identities`*, `T(0)`.

```
template<typename T>
struct number_one { see below };
```

3 *Value*: `T`'s neutral element for multiplication (IEV 102-01-19), if any.

4 *Default value*: If `T` models *`inferable-identities`*, `T(1)`.

9.5.3.4 Associated types [num.assoc.types]

```
template<typename T>
using number_difference_t = decltype(std::declval<const T>() - std::declval<const T>());
```

1 `number_difference_t<T>` represents `T`'s difference's (IEV 102-01-17) type, if any.

```
template<typename T>
struct vector_scalar {};
template<see below>
struct vector_scalar<see below> {
  using type = see below;
};
```

2 *Type*: Scalar (IEV 102-02-18) for the vector space (IEV 102-03-01) `T`.

3 Each row of Table 5 denotes a specialization.

Table 5: Specializations of `vector_scalar` [tab:num.scalar]

template-parameter-list	*template-argument-list*	*Type*
`std::integral T`	`T`	`T`
`std::floating_point T`	`T`	`T`
`class T`	`std::complex<T>`	`T`
`class T, class U`	`std::chrono::duration<T, U>`	`T`

9.5.4 Concepts [num.concepts]

```
template<typename T>
concept number = enable_number_v<T> && std::regular<T>;

template<typename T, typename U>
concept common_number_with =
  number<T> && number<U> && std::common_with<T, U> && number<std::common_type_t<T, U>>;

template<typename T>
concept ordered_number = number<T> && std::totally_ordered<T>;

template<class T>
concept number_line =
  ordered_number<T> &&
  requires(T& v) {
    number_one_v<number_difference_t<T>>;
    { ++v } -> std::same_as<T&>;
    { --v } -> std::same_as<T&>;
    { v++ } -> std::same_as<T>;
    { v-- } -> std::same_as<T>;
  };
```

1 Let `v` and `u` be equal objects of type `T`, and `one` be the value `number_one_v<number_difference_t<T>>`. `T` models `number_line` only if

(1.1) — The expressions `++v` and `v++` have the same domain (C++ [concepts.equality]).

(1.2) — The expressions `--v` and `v--` have the same domain.

(1.3) — If `v` is incrementable (C++ [iterator.concept.winc]), then both `++v` and `v++` add `one` to `v`, and the following expressions all equal `true`:

(1.3.1) — `v++ == u`,

(1.3.2) — `((void)v++, v) == ++u`, and

(1.3.3) — `std::addressof(++v) == std::addressof(v)`.

(1.4) — If `v` is decrementable (C++ [range.iota.view]), then both `--v` and `v--` subtract `one` from `v`, and the following expressions all equal `true`:

(1.4.1) — `v-- == u`,

(1.4.2) — `((void)v--, v) == --u`, and

(1.4.3) — `std::addressof(--v) == std::addressof(v)`.

```
template<class T, class U> concept addition-with =
  number<T> &&
  number<U> &&
  requires(const T& c, const U& d) {
    { c + d } -> common_number_with<T>;
    { d + c } -> common_number_with<T>;
  };

template<class T, class U> concept compound-addition-with =
  addition-with<T, U> &&
  requires(T& l, const U& d) {
    { l += d } -> std::same_as<T&>;
  };

template<class T, class U> concept subtraction-with =
  addition-with<T, U> &&
  requires(const T& c, const U& d) {
    { c - d } -> common_number_with<T>;
  };
```

```
template<class T, class U> concept compound-subtraction-with =
  subtraction-with<T, U> &&
  compound-addition-with<T, U> &&
  requires(T& l, const U& d) {
    { l -= d } -> std::same_as<T&>;
  };

template<class T, class U, class V> concept multiplication-with =
  number<T> &&
  number<U> &&
  number<V> &&
  requires(const T& c, const U& u) {
    { c * u } -> common_number_with<V>;
  };

template<class T, class U> concept compound-multiplication-with =
  multiplication-with<T, U, T> &&
  requires(T& l, const U& u) {
    { l *= u } -> std::same_as<T&>;
  };

template<class T, class U> concept division-with =
  multiplication-with<T, U, T> &&
  multiplication-with<U, T, T> &&
  requires(const T& c, const U& u) {
    { c / u } -> common_number_with<T>;
  };

template<class T, class U> concept compound-division-with =
  division-with<T, U> &&
  compound-multiplication-with<T, U> &&
  requires(T& l, const U& u) {
    { l /= u } -> std::same_as<T&>;
  };

template<class T, class U> concept modulo_with =
  number<T> &&
  number<U> &&
  requires(const T& c, const U& u) {
    { c % u } -> common_number_with<T>;
  };

template<class T, class U> concept compound_modulo_with =
  modulo_with<T, U> &&
  requires(T& l, const U& u) {
    { l %= u } -> std::same_as<T&>;
  };
```

2 Let `q` be an object of type `T`, and `r` be an object of type `U`.

3 For *`addition-with`* and *`compound-addition-with`*, let E be the expression `q + r` or `r + q`, and `q += r`, respectively, and let F be the addition (IEV 102-01-11) of the inputs to E.

4 For *`subtraction-with`* and *`compound-subtraction-with`*, let E be the expression `q - r` and `q -= r`, respectively, and let F be the subtraction (IEV 102-01-13) of the inputs to E.

5 For *`multiplication-with`* and *`compound-multiplication-with`*, let E be the expression `q * r` and `q *= r`, respectively, and let F be the multiplication (IEV 102-01-18) of the inputs to E.

6 For *`division-with`* and *`compound-division-with`*, let E be the expression `q / r` and `q /= r`, respectively, and let F be the division (IEV 102-01-21) of the inputs to E.

7 For `modulo_with` and `compound_modulo_with`, let E be the expression `q % r` and `q %= r`, respectively, and let F be the modulo (9.3.1) of the inputs to E.

8 `T` respectively models *`addition-with`*`<U>`, *`compound-addition-with`*`<U>`, *`subtraction-with`*`<U>`, *`compound-subtraction-with`*`<U>`, *`multiplication-with`*`<U, V>`, *`compound-multiplication-with`*`<U>`,

division-with<U>, *compound-division-with*<U>, `modulo_with<U>`, and `compound_modulo_with<U>` only if, for each respective E, when the inputs to E are in the domain of E

(8.1) — E performs F in an unspecified set.

(8.2) — If the operator of E is an *assignment-operator*,

(8.2.1) — E maps the value of F to `q`, and the result of E is a reference to `q`, and

(8.2.2) — the result of E is the value of F mapped to the type of E otherwise.

```
template<typename T, typename U>
concept modulus_for = modulo_with<U, T>;
template<typename T, typename U>
concept compound_modulus_for = compound_modulo_with<U, T>;

template<class T> concept negative =
  compound-addition-with<T, T> &&
  requires(const T& c) {
    number_zero_v<T>;
    { -c } -> common_number_with<T>;
  };

template<class T> concept set_with_inverse =
  compound-multiplication-with<T, T> &&
  requires(const T& c) {
    { number_one_v<T> / c } -> std::common_with<T>;
  };
```

9 Let `v` be an object of type `T`.

10 For `negative`, let E be the expression `-v`, and let F be the negative (IEV 102-01-14) of `v`.

11 For `set_with_inverse`, let E be the expression `number_one_v<T> / v`, and let F be the inverse (IEV 102-01-24) of `v`.

12 `T` respectively models `negative<U>` and `set_with_inverse<U>` only if, for the respective E, when `v` is in the domain of E

(12.1) — E performs F in an unspecified set.

(12.2) — The result of E is the value of F mapped to the type of E.

```
template<typename T, typename U>
concept point_space_for =
  subtraction-with<T, U> && negative<U> && common_number_with<number_difference_t<T>, U>;
```

13 Let `q` and `r` be objects of type `T`.

14 Let E be the expression `q - r`, and let F be the subtraction of the inputs to E.

15 `T` models `point_space_for<U>` only if, when the inputs to E are in the domain of E

(15.1) — E performs F in an unspecified set.

(15.2) — The result of E is the value of F mapped to the type of E.

```
template<typename T, typename U>
concept compound_point_space_for = point_space_for<T, U> && compound-subtraction-with<T, U>;
template<typename T>
concept point_space = compound_point_space_for<T, number_difference_t<T>>;
```

16 [*Note 1*: The `point_space` concept is modeled by types that behave similarly to `std::chrono::sys_seconds`. — *end note*]

```
template<typename T, typename U>
concept vector_space_for = point_space_for<U, T>;
template<typename T, typename U>
concept compound_vector_space_for = compound_point_space_for<U, T>;

template<typename T>
concept weak-scalar =
  common_number_with<T, number_difference_t<T>> && point_space<T> && negative<T>;
```

```
template<typename T, typename U>
concept scales-with = common_number_with<U, vector_scalar_t<T>> && weak-scalar<U> &&
                      multiplication-with<T, U, T> && set_with_inverse<U>;
template<typename T, typename U>
concept compound-scales-with = scales-with<T, U> && compound-multiplication-with<T, U>;

template<typename T, typename U>
concept scalar_for = scales-with<U, T>;
template<typename T, typename U>
concept field_for = scalar_for<T, U> && division-with<U, T>;
template<typename T, typename U>
concept compound_scalar_for = compound-scales-with<U, T>;
template<typename T, typename U>
concept compound_field_for = compound_scalar_for<T, U> && compound-division-with<U, T>;

template<typename T>
concept vector_space = point_space<T> && compound-scales-with<T, vector_scalar_t<T>>;
template<typename T>
concept f_vector_space = vector_space<T> && compound-division-with<T, vector_scalar_t<T>>;
```

17 [*Note 2*: The `f_vector_space` concept is modeled by types that behave similarly to `std::chrono::seconds`. — *end note*]

```
template<typename T>
concept field_number = f_vector_space<T> && compound-scales-with<T, T>;

template<typename T>
concept field_number_line = field_number<T> && number_line<T>;

template<typename T>
concept scalar_number = field_number<T> && (field_number_line<T> || enable_complex_number_v<T>);
```

18 [*Note 3*: The `field_number` concept is modeled by types that behave similarly to `std::complex<double>`. It represents an approximation of a field, see IEV 102-02-18, Note 2 to entry. — *end note*]

19 [*Note 4*: The `field_number_line` concept is modeled by types that behave similarly to `double`. — *end note*]

20 [*Note 5*: `scalar_number` represents an approximation of a scalar number (IEV 102-02-18). — *end note*]

Cross references

Each clause and subclause label is listed below along with the corresponding clause or subclause number and page number, in alphabetical order by label.

Index

Index of library headers

The bold page number for each entry refers to the page where the synopsis of the header is shown.

Index of library names

Index of library concepts

The bold page number for each entry is the page where the concept is defined. Other page numbers refer to pages where the concept is mentioned in the general text.