of these concepts has been agreed upon by the SeqAn team but not yet fully implemented in the current master branch. This is the specification that I propose.

`seqan3::aligned_range<T>` requires:

- `T` must also model at least `seqan3::forward_range` so it can safely be iterated over multiple times.
- `seqan3::is_gap(r, it)` must be valid, where `r` is of type `T` and `it` is an iterator of that range.
  - `seqan3::is_gap` is a customisation point object defined for this purpose.
  - It returns `true` or `false` depending on whether the element pointed to by `it` represents a gap or not.
  - The CPO has a default implementation for ranges whose alphabet is comparable with `seqan3::gap`.
  - Custom implementations / specialisations can be given in the typical ways (see subsection 6.1.2).

This concept is met automatically by e.g. `std::vector<seqan3::gapped<seqan3::dna4>>`, but it also allows adapting ranges where the gap information cannot be queried directly or that use an entirely different alphabet to indicate gap symbols.

`seqan3::writable_aligned_range<T>` requires:

- `T` must also model `seqan3::aligned_range`.
- `seqan3::insert_gaps(r, it, n)` must be valid, where `r` is of type `T`, `it` is an iterator of that range and `n` is of type `size_t`.
  - `seqan3::insert_gaps` is a customisation point object defined for this purpose.
  - It inserts `n` gap symbols into `r` before `it`. The third parameter is optional, by default 1 gap is inserted.
  - The CPO has a default implementation for ranges that provide a member invocable in the following way: `.insert(it, n, seqan3::gap{})`. If `n` equals 1, `it` equals the end of the range and the following statement is valid, it is chosen instead: `.push_back(seqan3::gap{})`.
  - Custom implementations / specialisations can be given in the typical ways (see subsection 6.1.2).
- `seqan3::remove_gaps(r, it, sen)` must be valid, where `r` is of type `T`, `it` is an iterator of that range and `sen` is a sentinel or iterator of that range.
  - `seqan3::remove_gaps` is a customisation point object defined for this purpose.
  - It removes all gap symbols between `it` and `sen` (not including `sen`). The third parameter is optional, by default only a single gap is removed (if present). The number of removed gaps is returned.
  - The CPO has a default implementation for all ranges that model `seqan3::aligned_range`, it calls `std::ranges::remove_if(it, sen, seqan3::is_gap)`. If `std::next(it)` equals the end of the range, `seqan3::is_gap(*it)` is `true` and the following statement is valid, it is chosen instead: `r.pop_back()`.
  - Custom implementations / specialisations can be given in the typical ways (see subsec-

tion 6.1.2).

This definition implies that standard containers and types modelled after them automatically satisfy the requirements of `seqan3::writable_aligned_range` if their value type is `seqan3::gapped</**/>` (or a more nested type including `seqan3::gap`).

In addition to these concepts a third two-parameter concepts is defined.

`resettable_aligned_range<aligned_t, unaligned_t>` requires:

- `aligned_t` must also model `seqan3::writable_aligned_range`.
- `seqan3::reset_aligned_range(r, src)` must be valid where `r` is of type `aligned_t` and `src` is of type `unaligned_t`.
  - `seqan3::reset_aligned_range` is a CPO defined for this purpose.
  - It "assigns" the `src` to the `r`. The exact semantics depend on the `aligned_t`, but it is assumed that this CPO clears all gaps from `r` and that `std::ranges::equal(r, s)` is valid and returns true after the CPO is invoked.
  - The CPO defaults to `r.reset(src)` if that is valid. For SeqAn3' gap decorators over the `unaligned_t` this resets the internal pointer to the newly specified range (in addition to clearing gap information).
  - The CPO has an implementation for all types that are `seqan3::back_insertable_with` the source alphabet type (containers). `r` is cleared and elements are copied from the source and converted.
  - Custom implementations / specialisations can be given in the typical ways (see subsection 6.1.2).

The default implementations cover assigning e.g. `std::vector<seqan3::dna4>` to either `std::vector<seqan3::gapped<seqan3` (clear, copy) or `seqan3::gap_decorator<std::vector<seqan3::dna4>>` (clear, reset).

9.1.2. Gap decorators

As the default implementations for the CPOs have already indicated, the simplest way to make an aligned range from a non-aligned range is to create a `std::vector<seqan3::gapped<T>` where `T` is the alphabet of the original range and copy all elements into this vector interspersing gap symbols where needed. This kind of aligned range has the best possible read performance and especially for small ranges, the cost of random inserts is still tolerable. Since the entire original range needs to be copied, the space overhead, however, is very noticeable – assuming that the original range needs to be kept in memory, too.[2]

To reduce space and (potentially) increase random write performance, various adaptors can be devised that only hold a pointer to the original range and store the gap information in separate data structures. Various approaches and their algorithmic complexities are shown in Table 9.3. Here are brief descriptions:

Vector of gapped The full-fledged vector over a gapped alphabet. Fast, but huge size overhead.

Sparse bitvector An adaptor that holds an `sdsl::sd_vector` of the aligned sequence's size (n+g) with gap positions indicated by a 1. Useful if $^g/_k$ is small and no modifications happen after

[2]For small ranges this might still be worth it!