

Package ‘glyrepr’

February 13, 2026

Title Representation for Glycan Compositions and Structures

Version 0.10.1

Description Computational representations of glycan compositions and structures, including details such as linkages, anomers, and substituents. Supports varying levels of monosaccharide specificity (e.g., ``Hex" or ``Gal") and ambiguous linkages. Provides robust parsing and generation of IUPAC-condensed structure strings. Optimized for vectorized operations on glycan structures, with efficient handling of duplications. As the cornerstone of the glycoverse ecosystem, this package delivers the foundational data structures that power glycomics and glycoproteomics analysis workflows.

License MIT + file LICENSE

Suggests testthat (>= 3.0.0), patrick, tibble, knitr, rmarkdown, tictoc, lobstr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

URL <https://glycoverse.github.io/glyrepr/>,
<https://github.com/glycoverse/glyrepr>

Imports checkmate, cli, dplyr, furr, future, glue, igraph, magrittr, pillar, purrr, rlang, rstackdeque, stringr, vctrs (>= 0.6.5)

VignetteBuilder knitr

BugReports <https://github.com/glycoverse/glyrepr/issues>

NeedsCompilation no

Author Bin Fu [aut, cre, cph] (ORCID: <<https://orcid.org/0000-0001-8567-2997>>)

Maintainer Bin Fu <23110220018@m.fudan.edu.cn>

Repository CRAN

Date/Publication 2026-02-13 07:00:36 UTC

Contents

as_glycan_composition	2
as_glycan_structure	3
available_monosaccharides	4
available_substituents	5
convert_to_generic	5
count_mono	7
get_anomer	8
get_mono_type	8
get_structure_graphs	10
get_structure_level	10
glycan_composition	11
glycan_structure	12
has_linkages	15
is_known_monosaccharide	16
n_glycan_core	17
possible_linkages	18
reduce_structure_level	19
remove_linkages	20
remove_substituents	21
simap	21
smap	23
smap2	25
smap_predicates	27
smap_unique	28
smap	29
structure_to_iupac	31
valid_linkages	32
Index	34

as_glycan_composition *Convert to Glycan Composition*

Description

Converts an object to a glycan composition using vctrs casting framework. This function provides a convenient way to convert various input types to `glycan_composition()`.

Usage

```
as_glycan_composition(x)
```

Arguments

- x
- An object to convert to a glycan composition. Supported inputs include:
- Named integer vectors or lists of named integer vectors
 - Character vectors with composition strings (e.g., "Hex(5)HexNAc(2)")
 - glyrepr_structure objects (counts both monosaccharides and substituents)
 - Existing glyrepr_composition objects (returned as-is)

Details

This function uses the vctrs casting framework for type conversion. When converting from glycan structures, both monosaccharides and substituents are counted. Substituents are extracted from the sub attribute of each vertex in the structure. For example, a vertex with sub = "3Me" contributes one "Me" substituent to the composition.

Value

A glyrepr_composition object.

Examples

```
# From a single named vector
as_glycan_composition(c(Hex = 5, HexNAc = 2))

# From a list of named vectors
as_glycan_composition(list(c(Hex = 5, HexNAc = 2), c(Hex = 3, HexNAc = 1)))

# From a character vector of Byonic composition strings
as_glycan_composition(c("Hex(5)HexNAc(2)", "Hex(3)HexNAc(1)"))

# From a character vector of simple composition strings
as_glycan_composition(c("H5N2", "H5N4S1F1"))

# From an existing composition (returns as-is)
comp <- glycan_composition(c(Hex = 5, HexNAc = 2))
as_glycan_composition(comp)

# From a glycan structure vector
strucs <- c(n_glycan_core(), o_glycan_core_1())
as_glycan_composition(strucs)
```

as_glycan_structure *Convert to Glycan Structure Vector*

Description

Convert an object to a glycan structure vector.

Usage

```
as_glycan_structure(x)
```

Arguments

x An object to convert to a glycan structure vector. Can be an igraph object, a list of igraph objects, a character vector of IUPAC-condensed strings, or an existing glyrepr_structure object.

Value

A glyrepr_structure object.

Examples

```
library(igraph)

# Convert a single igraph
graph <- make_graph(~ 1--2)
V(graph)$mono <- c("GlcNAc", "GlcNAc")
V(graph)$sub <- ""
E(graph)$linkage <- "b1-4"
graph$anomer <- "a1"
as_glycan_structure(graph)

# Convert a list of igraphs
o_glycan_vec <- o_glycan_core_1()
o_glycan_graph <- get_structure_graphs(o_glycan_vec)
as_glycan_structure(list(graph, o_glycan_graph))

# Convert a character vector of IUPAC-condensed strings
as_glycan_structure(c("GlcNAc(b1-4)GlcNAc(b1-", "Man(a1-2)GlcNAc(b1-"))
```

available_monosaccharides

Get Available Monosaccharides

Description

This function returns a character vector of monosaccharide names of the given type. See [get_mono_type\(\)](#) for monosaccharide types.

Usage

```
available_monosaccharides(mono_type = "all")
```

Arguments

mono_type A character string specifying the type of monosaccharides. Can be "all", "generic", or "concrete". Default is "all".

Value

A character vector of monosaccharide names.

Examples

```
available_monosaccharides()
```

available_substituents *Available Substituents*

Description

Get the available substituents for monosaccharides.

Usage

```
available_substituents()
```

Value

A character vector.

Examples

```
available_substituents()
```

convert_to_generic *Convert Monosaccharides to Generic Type*

Description

This function converts monosaccharide types of monosaccharide characters, glycan compositions, or glycan structures from concrete to generic type. This is a simplified version that only supports conversion from "concrete" to "generic" monosaccharides.

Usage

```
convert_to_generic(x)

## S3 method for class 'character'
convert_to_generic(x)

## S3 method for class 'glyrepr_structure'
convert_to_generic(x)

## S3 method for class 'glyrepr_composition'
convert_to_generic(x)
```

Arguments

x Either of these objects:

- A character of monosaccharide;
- A glycan composition vector ("glyrepr_composition" object);
- A glycan structure vector ("glyrepr_structure" object).

Value

A new object of the same class as x with monosaccharides converted to generic type.

Two types of monosaccharides

There are two types of monosaccharides:

- concrete: e.g. "Gal", "GlcNAc", "Glc", "Fuc", etc.
- generic: e.g. "Hex", "HexNAc", "HexA", "HexN", etc.

For the full list of monosaccharides, use [available_monosaccharides\(\)](#).

Examples

```
# Convert character vectors
convert_to_generic(c("Gal", "GlcNAc"))

# Convert glycan compositions
comps <- glycan_composition(
  c(Gal = 5, GlcNAc = 2),
  c(Glc = 5, GalNAc = 4, Fuc = 1)
)
convert_to_generic(comps)

# Convert glycan structures
strucs <- c(n_glycan_core(), o_glycan_core_1())
convert_to_generic(strucs)
```

count_mono	<i>Get the Number of Monosaccharides</i>
------------	--

Description

When mono is:

- NULL (default), returns the total number of monosaccharides and substituents.
- A string, returns the number of the specified monosaccharide or substituent.

Usage

```
count_mono(x, mono = NULL, include_subs = FALSE)
```

```
## S3 method for class 'glyrepr_composition'  
count_mono(x, mono = NULL, include_subs = FALSE)
```

```
## S3 method for class 'glyrepr_structure'  
count_mono(x, mono = NULL, include_subs = FALSE)
```

Arguments

x	A glycan composition (glyrepr_composition) or a glycan structure (glyrepr_structure) vector.
mono	The monosaccharide or substituent to count. A character scalar. If NULL (default), return the total number of monosaccharides.
include_subs	Whether to include substituents when mono is NULL. Default is FALSE.

Details

When mono is "generic" (e.g. "Hex", "HexNAc"), it counts all "concrete" monosaccharides that match. For example, "Hex" will count all Glc, Man, Gal, etc. When mono is "concrete" (e.g. "Gal", "GalNAc"), NA is returned when the composition is "generic".

Value

A numeric vector of the same length as x.

Examples

```
comp <- glycan_composition(c(Gal = 1, Man = 1, GalNAc = 1))  
count_mono(comp, "Hex")  
count_mono(comp, "Gal")  
  
struct <- as_glycan_structure("Gal(b1-3)GlcNAc(b1-4)Glc(a1-")  
count_mono(struct, "Gal")  
  
# Total number of monosaccharides
```

```
count_mono(comp)
```

```
get_anomer          Get the Anomeric information
```

Description

Get the Anomeric information

Usage

```
get_anomer(x)
```

Arguments

x A glycan structure vector (glyrepr_structure).

Value

a character vector of the anomeric information.

Examples

```
x <- n_glycan_core()
get_anomer(x)
```

```
get_mono_type      Get Monosaccharide Types
```

Description

This function determines the type of monosaccharides in character vectors, glycan compositions, or glycan structures. Supported types: "concrete" and "generic" (see details below).

Usage

```
get_mono_type(x)

## S3 method for class 'character'
get_mono_type(x)

## S3 method for class 'glyrepr_structure'
get_mono_type(x)

## S3 method for class 'glyrepr_composition'
get_mono_type(x)
```


Arguments

- x
- Either of these objects:
- A character vector of monosaccharide names;
 - A glycan composition vector ("glyrepr_composition" object);
 - A glycan structure vector ("glyrepr_structure" object).

Value

- For character input, returns a character vector of the same length as x.
- For glyrepr_structure and glyrepr_composition input, returns a character scalar.

Two types of monosaccharides

There are two types of monosaccharides:

- concrete: e.g. "Gal", "GlcNAc", "Glc", "Fuc", etc.
- generic: e.g. "Hex", "HexNAc", "HexA", "HexN", etc.

For the full list of monosaccharides, use [available_monosaccharides\(\)](#).

Special monosaccharides

Some monosaccharides are special in that they have no generic names in database or literature. For example, "Mur" is a rare monosaccharide that has no popular generic name. In glyrepr, we assign a "g" prefix to these monosaccharides as their generic names. This includes "gNeu", "gKdn", "gPse", "gLeg", "gAci", "g4eLeg", "gBac", "gKdo", "gMur". These names might only be meaningful inside glycoverse. Take care when you export results from glycoverse functions to other analysis tools.

See Also

[convert_to_generic\(\)](#)

Examples

```
# Character vector
get_mono_type(c("Gal", "Hex"))

# Glycan structures
get_mono_type(n_glycan_core(mono_type = "concrete"))
get_mono_type(n_glycan_core(mono_type = "generic"))

# Glycan compositions
comp <- glycan_composition(c(Glc = 2, GalNAc = 1))
get_mono_type(comp)
```

`get_structure_graphs` *Access Individual Glycan Structures*

Description

Extract individual glycan structure graphs from a glycan structure vector.

Usage

```
get_structure_graphs(x, return_list = NULL)
```

Arguments

<code>x</code>	A glycan structure vector.
<code>return_list</code>	If TRUE, always returns a list. If FALSE and <code>x</code> has a length of 1, return the igraph object directly. If not provided (default), FALSE when <code>x</code> has a length of 1 and TRUE otherwise.

Value

A list of igraph objects or an igraph object directly (see `return_list` parameter).

Examples

```
structures <- c(o_glycan_core_1(), n_glycan_core())
get_structure_graphs(structures)
get_structure_graphs(structures)
```

`get_structure_level` *Get the Structure Resolution Levels*

Description

Glycan structures can have four possible levels of resolution:

- "intact": All monosaccharides are concrete (e.g. "Man", "GlcNAc"), and all linkages are fully determined (e.g. "a2-3", "b1-4").
- "partial": All monosaccharides are concrete (e.g. "Man", "GlcNAc"), but some linkage information is missing (e.g. "a2-?").
- "topological": All monosaccharides are concrete (e.g. "Man", "GlcNAc"), but the linkage information is completely unknown ("??-?").
- "basic": All monosaccharides are generic (e.g. "Hex", "HexNAc"), and the linkage information is completely unknown ("??-?").

Note that in theory you can have a glycan with generic monosaccharides with all linkages determined. For example, "Hex(b1-3)HexNAc(a1-)" is a valid glycan structure. But in reality, this is almost impossible, because linkage information is far more difficult to acquire than monosaccharide information. This kind of glycan structure is also assigned to "basic" level.

Usage

```
get_structure_level(x)
```

Arguments

x A `glycan_structure()` vector.

Value

A character vector of the same length as x, containing the structure level for each element.

See Also

[has_linkages\(\)](#), [get_mono_type\(\)](#)

Examples

```
glycan <- as_glycan_structure("Gal(b1-3)GalNAc(a1-")
get_structure_level(glycan)
```

glycan_composition *Create a Glycan Composition*

Description

Create a glycan composition from a list of named integer vectors. Compositions can contain both monosaccharides and substituents.

Usage

```
glycan_composition(...)
```

```
is_glycan_composition(x)
```

Arguments

... Named integer vectors. Names are monosaccharides or substituents, values are numbers of residues. Monosaccharides and substituents can be mixed in the same composition.

x A list of named integer vectors.

Details

Compositions can contain:

- Monosaccharides: either generic (e.g., "Hex", "HexNAc") or concrete (e.g., "Glc", "Gal"). All monosaccharides in a composition vector must be of the same type.
- Substituents: e.g., "Me", "Ac", "S". These can be mixed with either generic or concrete monosaccharides.

Components are automatically sorted with monosaccharides first (according to their order in the monosaccharides table), followed by substituents (according to their order in available_substituents()).

Value

A glyrepr_composition object.

See Also

available_monosaccharides(), available_substituents()

Examples

```
# A vector with one composition (generic monosaccharides)
glycan_composition(c(Hex = 5, HexNAc = 2))
# A vector with multiple compositions
glycan_composition(c(Hex = 5, HexNAc = 2), c(Hex = 5, HexNAc = 4, dHex = 2))
# Residues are reordered automatically
glycan_composition(c(HexNAc = 1, Hex = 2))
# An example for generic monosaccharides
glycan_composition(c(Hex = 2, HexNAc = 1))
# An example for concrete monosaccharides
glycan_composition(c(Glc = 2, Gal = 1))
# Compositions with substituents
glycan_composition(c(Glc = 1, S = 1))
glycan_composition(c(Hex = 3, HexNAc = 2, Me = 1, Ac = 1))
# Substituents are sorted after monosaccharides
glycan_composition(c(S = 1, Gal = 1, Ac = 1, Glc = 1))
```

glycan_structure

Create a Glycan Structure Vector

Description

glycan_structure() creates an efficient glycan structure vector for storing and processing glycan molecular structures. The function employs hash-based deduplication mechanisms, making it suitable for glycoproteomics, glycomics analysis, and glycan structure comparison studies.

Usage

```
glycan_structure(...)  
  
is_glycan_structure(x)
```

Arguments

... igraph graph objects to be converted to glycan structures, or existing glycan structure vectors. Supports mixed input of multiple objects.

x An object to check or convert.

Value

A glyrepr_structure class glycan structure vector object.

Core Features

- **Efficient Storage:** Uses hash values of IUPAC codes for deduplication, avoiding redundant storage of identical glycan structures
- **Graph Model Representation:** Each glycan structure is represented as a directed graph where nodes are monosaccharides and edges are glycosidic linkages
- **Vectorized Operations:** Supports R's vectorized operations for batch processing of glycan data
- **Type Safety:** Built on the vctrs package, providing type-safe operations

Data Structure Overview

A glycan structure vector is a vctrs record with an additional S3 class glyrepr_structure. Therefore, `sloop::s3_class()` returns the class hierarchy `c("glyrepr_structure", "vctrs_rcrd")`.

Each glycan structure must satisfy the following constraints:

Graph Structure Requirements:

- Must be a directed graph with an outward tree structure (reducing end as root)
- Must have a graph attribute `anomer` in the format "a1" or "b1"
 - Unknown parts can be represented with "?", e.g., "?1", "a?", "??"

Node Attributes:

- `mono`: Monosaccharide names, must be known monosaccharide types
 - Generic names: Hex, HexNAc, dHex, NeuAc, etc.
 - Concrete names: Glc, Gal, Man, GlcNAc, etc.
 - Cannot mix generic and concrete names
 - NA values are not allowed
- `sub`: Substituent information
 - Single substituent format: "xY" (x = position, Y = substituent name), e.g., "2Ac", "3S"
 - Multiple substituents separated by commas and ordered by position, e.g., "3Me,4Ac", "2S,6P"

- No substituents represented by empty string ""

Edge Attributes:

- `linkage`: Glycosidic linkage information in format "a/bX-Y"
 - Standard format: e.g., "b1-4", "a2-3"
 - Unknown positions allowed: "a1-?", "b?-3", "??-?"
 - Partially unknown positions: "a1-3/6", "a1-3/6/9"
 - NA values are not allowed

Node and Edge Order

The indices of vertices and linkages in a glycan correspond directly to their order in the IUPAC-condensed string, which is printed when you print a `glycan_structure()`. For example, for the glycan `Man(a1-3)[Man(a1-6)]Man(b1-4)GlcNAc(b1-4)GlcNAc(b1-`, the vertices are "Man", "Man", "Man", "GlcNAc", "GlcNAc", and the linkages are "a1-3", "a1-6", "b1-4", "b1-4".

Use Cases

- **Glycoproteomics Analysis**: Processing glycan structure information from mass spectrometry data
- **Glycomics Research**: Comparing glycan expression profiles across different samples or conditions
- **Structure-Function Analysis**: Studying relationships between glycan structures and biological functions
- **Database Queries**: Performing structure matching and searches in glycan databases

NA Support

Glycan structure vectors support NA values for representing missing or unknown structures:

- Create with `glycan_structure(NA)` or `glycan_structure(NULL)`
- Combine with valid structures: `c(struct1, NA, struct2)`
- Convert from character: `as_glycan_structure(c("Glc(a1-", NA))`
- `smap` functions skip NA elements gracefully
- `is.na()` returns TRUE for NA elements

Examples

```
library(igraph)

# Example 1: Create a simple glycan structure GlcNAc(b1-4)GlcNAc
graph <- make_graph(~ 1--2) # Create graph with two monosaccharides
V(graph)$mono <- c("GlcNAc", "GlcNAc") # Set monosaccharide types
V(graph)$sub <- "" # No substituents
E(graph)$linkage <- "b1-4" # b1-4 glycosidic linkage
graph$anomer <- "a1" # a anomeric carbon

# Create glycan structure vector
```

```
simple_struct <- glycan_structure(graph)
print(simple_struct)

# Example 2: Use predefined glycan core structures
n_core <- n_glycan_core() # N-glycan core structure
o_core1 <- o_glycan_core_1() # O-glycan Core 1 structure

# Example 3: Create complex structure with substituents
complex_graph <- make_graph(~ 1-+2-+3)
V(complex_graph)$mono <- c("GlcNAc", "Gal", "Neu5Ac")
V(complex_graph)$sub <- c("", "", "") # Add substituents as needed
E(complex_graph)$linkage <- c("b1-4", "a2-3")
complex_graph$anomer <- "b1"

complex_struct <- glycan_structure(complex_graph)
print(complex_struct)

# Example 4: Check if object is a glycan structure
is_glycan_structure(simple_struct) # TRUE
is_glycan_structure(graph)         # FALSE
```

has_linkages

Determine if a Glycan Structure has Linkages

Description

Unknown linkages in a glycan structure are represented by "??-?". Also, a linkage can be partially known (e.g. "a?-?"). This function checks if a glycan structure has linkages, in a strict or lenient way.

Usage

```
has_linkages(glycan, strict = FALSE)
```

Arguments

glycan A `glycan_structure()` vector.

strict A logical value.

- If FALSE (default), a glycan is considered to have linkages if any linkage is partially known (not "??-?").
- If TRUE, a glycan is considered to have linkages only if all linkages are fully determined (no "?" in the linkage).

Value

A logical vector indicating if each glycan structure has linkages.

See Also

[remove_linkages\(\)](#), [possible_linkages\(\)](#)

Examples

```
glycan <- o_glycan_core_1(linkage = TRUE)
has_linkages(glycan)
print(glycan)
```

```
glycan <- remove_linkages(glycan)
has_linkages(glycan)
print(glycan)
```

```
glycan <- as_glycan_structure("Gal(b1-?)GalNAc(a1-")
has_linkages(glycan)
has_linkages(glycan, strict = TRUE)
```

is_known_monosaccharide

Check if a Monosaccharide is Known

Description

This function checks if a vector of monosaccharide names are known.

Usage

```
is_known_monosaccharide(mono)
```

Arguments

mono A character vector of monosaccharide names.

Value

A logical vector.

Examples

```
is_known_monosaccharide(c("Gal", "Hex"))
is_known_monosaccharide(c("X", "Hx", "Nac"))
```

n_glycan_core *Example Glycan Structures*

Description

Create example glycan structures for testing and demonstration. Includes **N-glycan core** and **O-glycan core 1** and **core 2**.

Usage

```
n_glycan_core(linkage = TRUE, mono_type = "concrete")
o_glycan_core_1(linkage = TRUE, mono_type = "concrete")
o_glycan_core_2(linkage = TRUE, mono_type = "concrete")
```

Arguments

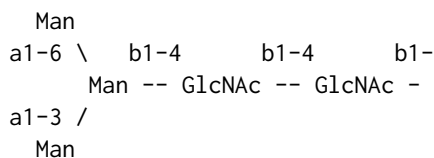
linkage	A logical indicating whether to include linkages (e.g. "b1-4"). Default is TRUE.
mono_type	A character string specifying the type of monosaccharides. Can be "generic" (Hex, HexNAc, dHex, NeuAc, etc.) or "concrete" (Man, Gal, HexNAc, Fuc, etc.). Default is "concrete".

Value

A glycan structure (igraph) object.

N-Glycan Core

N-Glycans are branched oligosaccharides that are bound, most commonly, via GlcNAc to an Asn residue of the protein backbone. A common motif of all N-glycans is the **chitobiose core**, composed of three mannose and two GlcNAc moieties, which is commonly attached to the protein backbone via GlcNAc. The mannose residue is branched and connected via a1,3- and a1,6-glycosidic linkages to the two other mannose building blocks.



O-Glycan Core

O-Glycans are highly abundant in extracellular proteins. Generally, O-glycans are extended following four major core structures: **core 1**, **core 2**, core 3, and core 4. The first two are by far the most common core structures in O-glycosylation and are found throughout the body.

core 1:

```

      a1-
    GalNAc -
  / b1-3
Gal

```

core 2:

```

GlcNAc
  \ b1-6 a1-
  GalNAc -
  / b1-3
Gal

```

Examples

```

print(n_glycan_core(), verbose = TRUE)
print(o_glycan_core_1(), verbose = TRUE)

```

possible_linkages *Generate Possible Linkages*

Description

Given an obscure linkage format (having "?", e.g. "a2-?"), this function generates all possible linkages based on the format. See [valid_linkages\(\)](#) for details.

The ranges of possible anomers, first positions, and second positions can be specified using `anomer_range`, `pos1_range`, and `pos2_range`.

Usage

```

possible_linkages(
  linkage,
  anomer_range = c("a", "b"),
  pos1_range = 1:2,
  pos2_range = 1:9,
  include_unknown = FALSE
)

```

Arguments

<code>linkage</code>	A linkage string.
<code>anomer_range</code>	A character vector of possible anomers. Default is <code>c("a", "b")</code> .
<code>pos1_range</code>	A numeric vector of possible first positions. Default is <code>1:2</code> .
<code>pos2_range</code>	A numeric vector of possible second positions. Default is <code>1:9</code> .
<code>include_unknown</code>	A logical value. If TRUE, "?" will be included. Default is FALSE.

Value

A character vector of possible linkages.

See Also

[has_linkages\(\)](#), [remove_linkages\(\)](#), [valid_linkages\(\)](#)

Examples

```
possible_linkages("a2-?")
possible_linkages("??-2")
possible_linkages("a1-3")
possible_linkages("a?-?", pos1_range = 2, pos2_range = c(2, 3))
possible_linkages("1-6", include_unknown = TRUE)
```

reduce_structure_level

Reduce a Glycan Structure to a Lower Resolution Level

Description

This function reduces a glycan structure from a higher resolution level to a lower resolution level (see [get_structure_level\(\)](#) for four possible levels of resolution). For example, it can reduce an "intact" structure to a "topological" structure, or a "partial" structure to a "basic" structure. One exception is that you can never reduce an "intact" structure to "partial" level, because the "partial" level is not deterministic.

Usage

```
reduce_structure_level(x, to_level)
```

Arguments

x	A glycan_structure() vector.
to_level	The resolution level to reduce to. Can be "basic" or "topological". Must be a lower resolution level than any structure in x ("intact" > "partial" > "topological" > "basic"). If to_level is the same as some structure in x, the result will be the same as the input. You can use get_structure_level() to check the structure levels of x.

Details

The logic is as follows:

- If to_level is "topological", this function calls [remove_linkages\(\)](#) to remove all linkages.
- If to_level is "basic", this function calls [remove_linkages\(\)](#) to remove all linkages, and [convert_to_generic\(\)](#) to convert all monosaccharides to generic.

Value

A `glycan_structure()` vector reduced to the given resolution level.

See Also

`get_structure_level()`

Examples

```
glycan <- as_glycan_structure("Gal(b1-3)GalNAc(a1-")
reduce_structure_level(glycan, to_level = "topological")
```

remove_linkages

Remove All Linkages from a Glycan

Description

This function replaces all linkages in a glycan structure with "??-", as well as the reducing end anomer with "??-".

Usage

```
remove_linkages(glycan)
```

Arguments

glycan A glyrepr_structure vector.

Value

A glyrepr_structure vector with all linkages removed.

Examples

```
glycan <- o_glycan_core_1(linkage = TRUE)
glycan
remove_linkages(glycan)
```

remove_substituents *Remove All Substituents from a Glycan*

Description

This function replaces all substituents in a glycan structure with empty strings.

Usage

```
remove_substituents(glycan)
```

Arguments

glycan A glyrepr_structure vector.

Value

A glyrepr_structure vector with all substituents removed.

Examples

```
(glycan <- o_glycan_core_1())
remove_substituents(glycan)
```

simap *Map Functions Over Glycan Structure Vectors with Indices*

Description

These functions apply a function to each unique structure in a glycan structure vector along with their corresponding indices, taking advantage of hash-based deduplication to avoid redundant computation. Similar to purrr imap functions, but optimized for glycan structure vectors.

Usage

```
simap(.x, .f, ...)

simap_vec(.x, .f, ..., .ptype = NULL)

simap_lgl(.x, .f, ...)

simap_int(.x, .f, ...)

simap_dbl(.x, .f, ...)
```

```
simap_chr(.x, .f, ...)
```

```
simap_structure(.x, .f, ...)
```

Arguments

<code>.x</code>	A glycan structure vector (<code>glyrepr_structure</code>).
<code>.f</code>	A function that takes an <code>igraph</code> object (from <code>.x</code>) and an index/name, returning a result. Can be a function, purrr-style lambda (<code>~ paste(.x, .y)</code>), or a character string naming a function.
<code>...</code>	Additional arguments passed to <code>.f</code> .
<code>.ptype</code>	A prototype for the return type (for <code>simap_vec</code>).

Details

These functions only compute `.f` once for each unique combination of structure and corresponding index/name, then map the results back to the original vector positions. This is much more efficient than applying `.f` to each element individually when there are duplicate structures.

IMPORTANT PERFORMANCE NOTE: Due to the inclusion of position indices, `simap` functions have **O(total_structures)** time complexity because each position creates a unique combination, even with identical structures.

Alternative: Consider `smap()` functions if position information is not required.

The index passed to `.f` is the position in the original vector (1-based). If the vector has names, the names are passed instead of indices.

Return Types:

- `simap()`: Returns a list with the same length as `.x`
- `simap_vec()`: Returns an atomic vector with the same length as `.x`
- `simap_lgl()`: Returns a logical vector
- `simap_int()`: Returns an integer vector
- `simap_dbl()`: Returns a double vector
- `simap_chr()`: Returns a character vector
- `simap_structure()`: Returns a new `glyrepr_structure` vector (`.f` must return `igraph` objects)

Value

- `simap()`: A list
- `simap_vec()`: An atomic vector of type specified by `.ptype`
- `simap_lgl()`: Returns a logical vector
- `simap_int()`: Returns an integer vector
- `simap_dbl()`: Returns a double vector
- `simap_chr()`: Returns a character vector
- `simap_structure()`: A new `glyrepr_structure` object

Examples

```
# Create structure vectors with duplicates
core1 <- o_glycan_core_1()
core2 <- n_glycan_core()
structures <- c(core1, core2, core1) # core1 appears twice

# Map a function that uses both structure and index
simap_chr(structures, function(g, i) paste0("Structure_", i, "_vcount_", igraph::vcount(g)))

# Use purrr-style lambda functions
simap_chr(structures, ~ paste0("Pos", .y, "_vertices", igraph::vcount(.x)))
```

smap

Map Functions Over Glycan Structure Vectors

Description

These functions apply a function to each unique structure in a glycan structure vector, taking advantage of hash-based deduplication to avoid redundant computation. Similar to purrr mapping functions, but optimized for glycan structure vectors.

Usage

```
smap(.x, .f, ..., .parallel = FALSE)

smap_vec(.x, .f, ..., .ptype = NULL, .parallel = FALSE)

smap_lgl(.x, .f, ..., .parallel = FALSE)

smap_int(.x, .f, ..., .parallel = FALSE)

smap_dbl(.x, .f, ..., .parallel = FALSE)

smap_chr(.x, .f, ..., .parallel = FALSE)

smap_structure(.x, .f, ..., .parallel = FALSE)
```

Arguments

<code>.x</code>	A glycan structure vector (glyrepr_structure).
<code>.f</code>	A function that takes an igraph object and returns a result. Can be a function, purrr-style lambda (<code>~ .x\$attr</code>), or a character string naming a function.
<code>...</code>	Additional arguments passed to <code>.f</code> .
<code>.parallel</code>	Logical; whether to use parallel processing. If FALSE (default), parallel processing is disabled. Set to TRUE to enable parallel processing.
<code>.ptype</code>	A prototype for the return type (for <code>smap_vec</code>).

Details

These functions only compute `.f` once for each unique structure, then map the results back to the original vector positions. This is much more efficient than applying `.f` to each element individually when there are duplicate structures.

Return Types:

- `smap()`: Returns a list with the same length as `.x`
- `smap_vec()`: Returns an atomic vector with the same length as `.x`
- `smap_lgl()`: Returns a logical vector
- `smap_int()`: Returns an integer vector
- `smap_dbl()`: Returns a double vector
- `smap_chr()`: Returns a character vector
- `smap_structure()`: Returns a new glycan structure vector (`.f` must return `igraph` objects)

Value

- `smap()`: A list
- `smap_vec()`: An atomic vector of type specified by `.ptype`
- `smap_lgl/int/dbl/chr()`: Atomic vectors of the corresponding type
- `smap_structure()`: A new `glyrepr_structure` object

Examples

```
# Create a structure vector with duplicates
core1 <- o_glycan_core_1()
core2 <- n_glycan_core()
structures <- c(core1, core2, core1) # core1 appears twice

# Map a function that counts vertices - only computed twice, not three times
smap_int(structures, igraph::vcount)

# Map a function that returns logical
smap_lgl(structures, function(g) igraph::vcount(g) > 5)

# Use purrr-style lambda functions
smap_int(structures, ~ igraph::vcount(.x))
smap_lgl(structures, ~ igraph::vcount(.x) > 5)

# Map a function that modifies structure (must return igraph)
add_vertex_names <- function(g) {
  if (!("name" %in% igraph::vertex_attr_names(g))) {
    igraph::set_vertex_attr(g, "name", value = paste0("v", seq_len(igraph::vcount(g))))
  } else {
    g
  }
}
smap_structure(structures, add_vertex_names)
```


smap2

*Map Functions Over Two Glycan Structure Vectors***Description**

These functions apply a function to each unique structure combination in two glycan structure vectors, taking advantage of hash-based deduplication to avoid redundant computation. Similar to purrr map2 functions, but optimized for glycan structure vectors.

Usage

```
smap2(.x, .y, .f, ..., .parallel = FALSE)

smap2_vec(.x, .y, .f, ..., .ptype = NULL, .parallel = FALSE)

smap2_lgl(.x, .y, .f, ..., .parallel = FALSE)

smap2_int(.x, .y, .f, ..., .parallel = FALSE)

smap2_dbl(.x, .y, .f, ..., .parallel = FALSE)

smap2_chr(.x, .y, .f, ..., .parallel = FALSE)

smap2_structure(.x, .y, .f, ..., .parallel = FALSE)
```

Arguments

<code>.x</code>	A glycan structure vector (<code>glyrepr_structure</code>).
<code>.y</code>	A vector of the same length as <code>.x</code> , or length 1 (will be recycled).
<code>.f</code>	A function that takes an <code>igraph</code> object (from <code>.x</code>) and a value (from <code>.y</code>) and returns a result. Can be a function, purrr-style lambda (<code>~ .x + .y</code>), or a character string naming a function.
<code>...</code>	Additional arguments passed to <code>.f</code> .
<code>.parallel</code>	Logical; whether to use parallel processing. If FALSE (default), parallel processing is disabled. Set to TRUE to enable parallel processing. See examples in smap for how to set up and use parallel processing.
<code>.ptype</code>	A prototype for the return type (for <code>smap2_vec</code>).

Details

These functions only compute `.f` once for each unique combination of structure and corresponding `.y` value, then map the results back to the original vector positions. This is much more efficient than applying `.f` to each element pair individually when there are duplicate structure-value combinations.

NA Handling: NA elements in `.x` are preserved in the output - the function is not applied to NA positions, and the corresponding results are set to NA.

Return Types:

- `smap2()`: Returns a list with the same length as `.x`
- `smap2_vec()`: Returns an atomic vector with the same length as `.x`
- `smap2_lgl()`: Returns a logical vector
- `smap2_int()`: Returns an integer vector
- `smap2_dbl()`: Returns a double vector
- `smap2_chr()`: Returns a character vector
- `smap2_structure()`: Returns a new glycan structure vector (`.f` must return `igraph` objects)

Value

- `smap2()`: A list
- `smap2_vec()`: An atomic vector of type specified by `.ptype`
- `smap2_lgl/int/dbl/chr()`: Atomic vectors of the corresponding type
- `smap2_structure()`: A new `glyrepr_structure` object

Examples

```
# Create structure vectors with duplicates
core1 <- o_glycan_core_1()
core2 <- n_glycan_core()
structures <- c(core1, core2, core1) # core1 appears twice
weights <- c(1.0, 2.0, 1.0) # corresponding weights

# Map a function that uses both structure and weight
smap2_dbl(structures, weights, function(g, w) igraph::vcount(g) * w)

# Use purrr-style lambda functions
smap2_dbl(structures, weights, ~ igraph::vcount(.x) * .y)

# Test with recycling (single weight for all structures)
smap2_dbl(structures, 2.5, ~ igraph::vcount(.x) * .y)

# Map a function that modifies structure based on second argument
# This example adds a graph attribute instead of modifying topology
add_weight_attr <- function(g, weight) {
  igraph::set_graph_attr(g, "weight", weight)
}
weights_to_add <- c(1.5, 2.5, 1.5)
smap2_structure(structures, weights_to_add, add_weight_attr)
```

Description

These functions test predicates on unique structures in a glycan structure vector, taking advantage of hash-based deduplication to avoid redundant computation. Similar to purrr predicate functions, but optimized for glycan structure vectors.

Usage

```
ssome(.x, .p, ...)
```

```
severy(.x, .p, ...)
```

```
snone(.x, .p, ...)
```

Arguments

<code>.x</code>	A glycan structure vector (glyrepr_structure).
<code>.p</code>	A predicate function that takes an igraph object and returns a logical value. Can be a function, purrr-style lambda (<code>~.x\$attr</code>), or a character string naming a function.
<code>...</code>	Additional arguments passed to <code>.p</code> .

Details

These functions only evaluate `.p` once for each unique structure, making them much more efficient than applying `.p` to each element individually when there are duplicate structures.

Return Values:

- `ssome()`: Returns TRUE if at least one unique structure satisfies the predicate
- `severy()`: Returns TRUE if all unique structures satisfy the predicate
- `snone()`: Returns TRUE if no unique structures satisfy the predicate

Value

A single logical value.

Examples

```
# Create a structure vector with duplicates
core1 <- o_glycan_core_1()
core2 <- n_glycan_core()
structures <- c(core1, core2, core1) # core1 appears twice

# Test if some structures have more than 5 vertices
```

```

ssome(structures, function(g) igraph::vcount(g) > 5)

# Test if all structures have at least 3 vertices
severy(structures, function(g) igraph::vcount(g) >= 3)

# Test if no structures have more than 20 vertices
snone(structures, function(g) igraph::vcount(g) > 20)

# Use purrr-style lambda functions
ssome(structures, ~ igraph::vcount(.x) > 5)
severy(structures, ~ igraph::vcount(.x) >= 3)
snone(structures, ~ igraph::vcount(.x) > 20)

```

smap_unique

Apply Function to Unique Structures Only

Description

Apply a function only to the unique structures in a glycan structure vector, returning results in the same order as the unique structures appear. This is useful when you need to perform expensive computations but only care about unique results.

Usage

```
smap_unique(.x, .f, ..., .parallel = FALSE)
```

Arguments

<code>.x</code>	A glycan structure vector (glyrepr_structure).
<code>.f</code>	A function that takes an igraph object and returns a result. Can be a function, purrr-style lambda (<code>~ .x\$attr</code>), or a character string naming a function.
<code>...</code>	Additional arguments passed to <code>.f</code> .
<code>.parallel</code>	Logical; whether to use parallel processing. If FALSE (default), parallel processing is disabled. Set to TRUE to enable parallel processing. See examples in smap for how to set up and use parallel processing.

Value

A list with results for each unique structure, named by their hash codes.

Examples

```

# Create a structure vector with duplicates
core1 <- o_glycan_core_1()
structures <- c(core1, core1, core1) # same structure 3 times

# Only compute once for the unique structure

```

```

unique_results <- smap_unique(structures, igrph::vcount)
length(unique_results) # 1, not 3

# Use purrr-style lambda
unique_results2 <- smap_unique(structures, ~ igrph::vcount(.x))
length(unique_results2) # 1, not 3

```

smap

Map Functions Over Glycan Structure Vectors and Multiple Arguments

Description

These functions apply a function to each unique structure in a glycan structure vector along with corresponding elements from multiple other vectors, taking advantage of hash-based deduplication to avoid redundant computation. Similar to purrr pmap functions, but optimized for glycan structure vectors.

Usage

```

smap(.l, .f, ..., .parallel = FALSE)

smap_vec(.l, .f, ..., .ptype = NULL, .parallel = FALSE)

smap_lgl(.l, .f, ..., .parallel = FALSE)

smap_int(.l, .f, ..., .parallel = FALSE)

smap_dbl(.l, .f, ..., .parallel = FALSE)

smap_chr(.l, .f, ..., .parallel = FALSE)

smap_structure(.l, .f, ..., .parallel = FALSE)

```

Arguments

.l	A list where the first element is a glycan structure vector (glyrepr_structure) and the remaining elements are vectors of the same length or length 1 (will be recycled).
.f	A function that takes an igrph object (from first element of .l) and values from other elements, returning a result. Can be a function, purrr-style lambda (~ .x + .y + .z), or a character string naming a function.
...	Additional arguments passed to .f.
.parallel	Logical; whether to use parallel processing. If FALSE (default), parallel processing is disabled. Set to TRUE to enable parallel processing. See examples in smap for how to set up and use parallel processing.
.ptype	A prototype for the return type (for smap_vec).

Details

These functions only compute `.f` once for each unique combination of structure and corresponding values from other vectors, then map the results back to the original vector positions.

NA Handling: NA elements in the first argument (glycan structure vector) are preserved in the output.

Time Complexity Performance:

Performance scales with unique combinations of all arguments rather than total vector length. When argument vectors are highly redundant, performance approaches $O(\text{unique_structures})$. Scaling factor shows time increase when vector size increases 20x.

Return Types:

- `smap()`: Returns a list with the same length as the input vectors
- `smap_vec()`: Returns an atomic vector with the same length as the input vectors
- `smap_lgl()`: Returns a logical vector
- `smap_int()`: Returns an integer vector
- `smap_dbl()`: Returns a double vector
- `smap_chr()`: Returns a character vector
- `smap_structure()`: Returns a new glycan structure vector (`.f` must return igraph objects)

Value

- `smap()`: A list
- `smap_vec()`: An atomic vector of type specified by `.ptype`
- `smap_lgl/int/dbl/chr()`: Atomic vectors of the corresponding type
- `smap_structure()`: A new `glyrepr_structure` object

Examples

```
# Create structure vectors with duplicates
core1 <- o_glycan_core_1()
core2 <- n_glycan_core()
structures <- c(core1, core2, core1) # core1 appears twice
weights <- c(1.0, 2.0, 1.0) # corresponding weights
factors <- c(2, 3, 2) # corresponding factors

# Map a function that uses structure, weight, and factor
smap_dbl(list(structures, weights, factors),
         function(g, w, f) igraph::vcount(g) * w * f)

# Use purrr-style lambda functions
smap_dbl(list(structures, weights, factors), ~ igraph::vcount(..1) * ..2 * ..3)

# Test with recycling
smap_dbl(list(structures, 2.0, 3), ~ igraph::vcount(..1) * ..2 * ..3)
```

structure_to_iupac	<i>Convert Glycan Structure to IUPAC-like Sequence</i>
--------------------	--

Description

Convert a glycan structure to a sequence representation in the form of mono(linkage)mono, with branches represented by square brackets []. The backbone is chosen as the longest path, and for branches, linkages are ordered lexicographically with smaller linkages on the backbone.

Usage

```
structure_to_iupac(glycan)
```

Arguments

glycan A glyrepr_structure vector.

Value

A character vector representing the IUPAC sequences.

Sequence Format

The sequence follows the format mono(linkage)mono, where:

- mono: monosaccharide name with optional substituents (e.g., Glc, GlcNAc, Glc3Me)
- linkage: glycosidic linkage (e.g., b1-4, a1-3)
- Branches are enclosed in square brackets []
- Substituents are appended directly to monosaccharide names (e.g., Glc3Me for Glc with 3Me substituent)

Backbone Selection

The backbone is selected as the longest path in the tree. For branches, the same rule applies recursively.

Linkage Comparison

Linkages are compared lexicographically:

1. First by anomeric configuration: ? > b > a
2. Then by first position: ? > numbers (numerically)
3. Finally by second position: ? > numbers (numerically)

Smaller linkages are placed on the backbone, larger ones in branches.

Examples

```

# Simple linear structure
structure_to_iupac(o_glycan_core_1())

# Branched structure
structure_to_iupac(n_glycan_core())

# Structure with substituents
graph <- igraph::make_graph(~ 1-+2)
igraph::V(graph)$mono <- c("Glc", "GlcNAc")
igraph::V(graph)$sub <- c("3Me", "6Ac")
igraph::E(graph)$linkage <- "b1-4"
graph$anomer <- "a1"
glycan <- glycan_structure(graph)
structure_to_iupac(glycan) # Returns "GlcNAc6Ac(b1-4)Glc3Me(a1-"

# Vectorized structures
structs <- c(o_glycan_core_1(), n_glycan_core())
structure_to_iupac(structs)

```

valid_linkages

Check if Linkages are Valid

Description

Valid linkages are in the form of "a1-2", "b1-4", "a?-1", etc. Specifically, the pattern is xy-z:

- x: the anomer, either "a", "b", or "?".
- y: the first position, either "1", "2" or "?".
- z: the second position, either a 1-9 digit or "?". Can also be multiple positions separated by "/", e.g. "1/2/3". "?" could not be used with "/".

Usage

```
valid_linkages(linkages)
```

Arguments

linkages A character vector of linkages.

Value

A logical vector.

Examples

```
# Valid linkages
valid_linkages(c("a1-2", "?1-4", "a?-1", "b?-?", "??-?", "a1/2-3"))

# Invalid linkages
valid_linkages(c("a1-2/?", "1-4", "a/b1-2", "c1-2", "a9-1"))
```

Index

as_glycan_composition, 2
as_glycan_structure, 3
available_monosaccharides, 4
available_monosaccharides(), 6, 9
available_substituents, 5

convert_to_generic, 5
convert_to_generic(), 9, 19
count_mono, 7

get_anomer, 8
get_mono_type, 8
get_mono_type(), 4, 11
get_structure_graphs, 10
get_structure_level, 10
get_structure_level(), 19, 20
glycan_composition, 11
glycan_composition(), 2
glycan_structure, 12
glycan_structure(), 11, 14, 15, 19, 20

has_linkages, 15
has_linkages(), 11, 19

is_glycan_composition
 (glycan_composition), 11
is_glycan_structure (glycan_structure),
 12
is_known_monosaccharide, 16

n_glycan_core, 17

o_glycan_core_1 (n_glycan_core), 17
o_glycan_core_2 (n_glycan_core), 17

possible_linkages, 18
possible_linkages(), 16

reduce_structure_level, 19
remove_linkages, 20
remove_linkages(), 16, 19

remove_substituents, 21

severy (smap_predicates), 27
simap, 21
simap_chr (simap), 21
simap_dbl (simap), 21
simap_int (simap), 21
simap_lgl (simap), 21
simap_structure (simap), 21
simap_vec (simap), 21
smap, 23, 25, 28, 29
smap2, 25
smap2_chr (smap2), 25
smap2_dbl (smap2), 25
smap2_int (smap2), 25
smap2_lgl (smap2), 25
smap2_structure (smap2), 25
smap2_vec (smap2), 25
smap_chr (smap), 23
smap_dbl (smap), 23
smap_int (smap), 23
smap_lgl (smap), 23
smap_predicates, 27
smap_structure (smap), 23
smap_unique, 28
smap_vec (smap), 23
snone (smap_predicates), 27
spmap, 29
spmap_chr (spmap), 29
spmap_dbl (spmap), 29
spmap_int (spmap), 29
spmap_lgl (spmap), 29
spmap_structure (spmap), 29
spmap_vec (spmap), 29
ssome (smap_predicates), 27
structure_to_iupac, 31

valid_linkages, 32
valid_linkages(), 18, 19