Package 'tidysq'

July 27, 2021

Type Package

Title Tidy Processing and Analysis of Biological Sequences

Version 1.1.2-1

Date 2021-07-12

Description A tidy approach to analysis of biological sequences. All processing and data-storage functions are heavily optimized to allow the fastest and most efficient data storage.

Depends R (>= 3.0.0)

Imports checkmate (>= 1.9.0), cli (>= 2.0.0), crayon (>= 1.3.4), dplyr (>= 1.0.2), pillar (>= 1.4.2), Rcpp (>= 1.0.1), tibble (>= 2.1.3), vctrs (>= 0.3.0)

Suggests ape (>= 5.3), bioseq (>= 0.1.2), Biostrings (>= 2.52.0), covr, knitr, lifecycle, purrr, seqinr (>= 3.4-5), spelling, rmarkdown, testthat (>= 3.0.0), withr (>= 2.2.0), rlang, mockr

License GPL (>= 2)

URL https://github.com/BioGenies/tidysq

BugReports https://github.com/BioGenies/tidysq/issues

SystemRequirements GNU make, C++17

NeedsCompilation yes

Repository CRAN

Encoding UTF-8

Language en-US

RoxygenNote 7.1.1

LinkingTo Rcpp, testthat

VignetteBuilder knitr

Config/testthat/edition 3

Author Dominik Rafacz [cre, aut] (<https://orcid.org/0000-0003-0925-1909>), Michal Burdukiewicz [aut] (<https://orcid.org/0000-0001-8926-582X>), Mateusz Bakala [aut], Leon Eyrich Jessen [ctb] (<https://orcid.org/0000-0003-2879-2559>),

Stefan Roediger [ctb] (<https://orcid.org/0000-0002-1441-6512>),
Jadwiga Slowik [ctb] (<https://orcid.org/0000-0003-3466-8933>),
Weronika Puchala [ctb] (<https://orcid.org/0000-0003-2163-1429>),
Katarzyna Sidorczuk [ctb],
Filip Pietluch [ctb],
Jaroslaw Chilimoniuk [ctb] (<https://orcid.org/0000-0001-5467-018X>)

Maintainer Dominik Rafacz <dominikrafacz@gmail.com>

Date/Publication 2021-07-27 13:20:02 UTC

R topics documented:

`tidysq-package` *tidysq: tidy analysis of biological sequences*

Description

The `tidysq` package is a toolbox for the analysis of biological sequences in a tidy way.

Author(s)

Michal Burdukiewicz, Dominik Rafacz, Mateusz Bąkała, Leon Eyrich Jessen

---

`==.sq` *Compare sq objects*

Description

Compares input `sq` object with either another `sq` object or `character` vector.

Usage

```
## S3 method for class 'sq'
e1 == e2
```

Arguments

`e1`	[`sq`] An object this comparison is applied to.
`e2`	[`sq` || `character`] An object to compare with `x1`.

Details

`` `==` `` compares compatible object for equality of their respective sequences. Objects are considered compatible, when either both have same length or one of them is a scalar value (i.e. a vector of length 1). Moreover, not every `e1` sq type can be compared to any `e2` sq type.

To see which types are compatible, see Details of `sq-concatenate`.

`` `==` `` returns logical vector, where each element describes whether elements at position `n` of both `e1` and `e2` are equal in meaning (that is, they may be represented differently, but their biological interpretation must be identical). If one of compared objects is a scalar, then said logical vector describes comparison for each element of the other, longer vector.

Value

A `logical` vector indicating on which positions these objects are equal.

See Also

Functions from utility module: get_sq_lengths(), is.sq(), sqconcatenate, sqextract

Examples

```
# Creating objects to work on:
sq_dna_1 <- sq(c("ACTGCTG", "CTTAGA", "CCCT", "CTGAATGT"),
               alphabet = "dna_bsc")
sq_dna_2 <- sq(c("ACTGCTG", "CTTAGA", "CCCT", "CTGAATGT"),
               alphabet = "dna_bsc")
sq_dna_3 <- sq(c("ACTGCTG", "CTTAGA", "GGAA"),
               alphabet = "dna_bsc")
sq_dna_4 <- sq(c("ACTGCTG", "CTTAGA", "CCCT", "GTNANN"),
               alphabet = "dna_ext")
sq_ami_1 <- sq(c("ACTGCTG", "NIKAAR", "CCCT", "CTGAATGT"),
               alphabet = "ami_bsc")
sq_unt <- sq(c("AHSNLVSCTK$SH%&VS", "YQTVKA&#BSKJGY",
               "CCCT", "AVYI#VSV&*DVGDJCFA"))

# Comparing sq object with an object of the same length:
sq_dna_1 == sq_dna_2
sq_dna_1 == c("ACTGCTG", "CTTAGA", "CCCT", "CTGAATGT")

# Cannot compare sq objects of different lengths:
## Not run:
sq_dna_1 == sq_dna_3
sq_dna_1 == c("AAA", "CCC")

## End(Not run)

# Unless comparing sq object with scalar value:
sq_dna_1 == "CTTAGA"

# It's possible to compare basic and extended types:
sq_dna_1 == sq_dna_4

# Mixing DNA, RNA and amino acid types throws an error, however:
## Not run:
sq_dna_1 == sq_ami_1

## End(Not run)

# On the other hand, unt sq is acceptable everywhere:
sq_dna_1 == sq_unt
sq_dna_4 == sq_unt
sq_ami_1 == sq_unt
```

---

alphabet *Get alphabet of given sq object.*

Description

Returns `alphabet` attribute of an object.

Usage

```
alphabet(x)
```

Arguments

`x`	[`sq`] An object to extract alphabet from.

Details

Each `sq` object have an **alphabet** associated with it. Alphabet is a set of possible **letters** that can appear in sequences contained in object. Alphabet is kept mostly as a character vector, where each element represents one **letter**.

`sq` objects of type **ami**, **dna** or **rna** have fixed, predefined alphabets. In other words, if two `sq` objects have exactly the same type - **ami_bsc**, **dna_ext**, **rna_bsc** or any other combination - they are ensured to have the same alphabet.

Below are listed alphabets for these types:

- **ami_bsc** - ACDEFGHIKLMNPQRSTVWY-*
- **ami_ext** - ABCDEFGHIJKLMNOPQRSTUVWXYZ-*
- **dna_bsc** - ACGT-
- **dna_ext** - ACGTWSMKRYBDHVN-
- **rna_bsc** - ACGU-
- **rna_ext** - ACGUWSMKRYBDHVN-

Other types of `sq` objects are allowed to have different alphabets. Furthermore, having an alphabet exactly identical to one of those above does not automatically indicate that the type of the sequence is one of those - e.g., there might be an **atp** `sq` that has an alphabet identical to **ami_bsc** alphabet. To set the type, one should use the `typify` or `` `sq_type<-` `` function.

The purpose of co-existence of **unt** and **atp** alphabets is the fact that although there is a standard for format of *fasta* files, sometimes there are other types of symbols, which do not match the standard. Thanks to these types, tidysq can import files with customized alphabets. Moreover, the user may want to group amino acids with similar properties (e.g., for machine learning) and replace the standard alphabet with symbols for whole groups. To check details, see `read_fasta`, `sq` and `substitute_letters`.

Important note: in **atp** alphabets there is a possibility of letters appearing that consist of more than one character - this functionality is provided in order to handle situations like post-translational modifications, (e.g., using "`mA`" to indicate methylated alanine).

Important note: alphabets of **atp** and **unt** `sq` objects are case sensitive. Thus, in their alphabets both lowercase and uppercase characters can appear simultaneously and they are treated as different letters. Alphabets of **dna**, **rna** and **ami** types are always uppercase and all functions converts other

parameters to uppercase when working with **dna**, **rna** or **ami** - e.g. %has% operator converts lower letters to upper when searching for motifs in **dna**, **rna** or **ami** object.

Important note: maximum length of an alphabet is **30 letters**. The user is not allowed to read fasta files or construct sq objects from character vectors that have more than 30 distinct characters in sequences (unless creating **ami**, **dna** or **rna** objects with ignore_case parameter set equal to TRUE).

Value

A character vector of letters of the alphabet.

See Also

sq class

Functions from alphabet module: get_standard_alphabet()

---

as.character.sq *Convert sq object into character vector*

Description

Coerces sequences from an sq object to character vector of sequences.

Usage

```
## S3 method for class 'sq'
as.character(x, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

x	[sq] An object this function is applied to.
...	further arguments to be passed from or to other methods.
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

Details

This method for sq class allows converting sequences from the sq object into a character vector of length equal to the length of input. Each element of resulting vector is a separate sequence. All attributes of the input sq are lost during the conversion to character vector.

Value

A character vector where each element represents the content of respective sequence in input sq object.

See Also

Functions from output module: as.matrix.sq(), as.sq(), export_sq(), write_fasta()

Examples

```
# Creating an object to work on:
sq_dna <- sq(c("CTGAATGCAGTACCGTAAT", "ATGCCGTAAATGCCAT", "CAGACCANNNATAG"),
             alphabet = "dna_ext")

# Converting sq object into a character vector:
as.character(sq_dna)
```

---

as.matrix.sq *Convert sq object into matrix*

Description

Coerces sequences from a sq object to a matrix, in which rows correspond to sequences and columns to positions.

Usage

```
## S3 method for class 'sq'
as.matrix(x, ...)
```

Arguments

x	[sq] An object this function is applied to.
...	further arguments to be passed from or to other methods.

Details

This method for class sq allows converting sequences from the sq object into a matrix. Each row corresponds to the separate sequence from the sq object, whereas each column indicates a single position within a sequence. Dimensions of matrix are determined by the number of sequences (rows) and the length of the longest sequence (columns). If length of a sequence is smaller than the length of the longest sequence, the remaining columns are filled with NA. All attributes of the input sq are lost during the conversion to matrix.

Value

A matrix with number of rows the same as number of sequences and number of columns corresponding to the length of the longest sequence in the converted sq object.

See Also

Functions from output module: as.character.sq(), as.sq(), export_sq(), write_fasta()

Examples

```
# Creating objects to work on:
sq_dna <- sq(c("CGATAGACA", "TGACAAAAC", "GTGACCGTA"),
             alphabet = "dna_bsc")
sq_rna <- sq(c("CUGAAUGCAGUACCGUAAU", "AUGCCGUAAAUGCCAU", "CAGACCANNNAUAG"),
             alphabet = "rna_ext")

# Sequences of the same lengths can be converted easily:
as.matrix(sq_dna)

# Sequences that differ in length are filled with NA to the maximum length:
as.matrix(sq_rna)
```

---

as.sq *Convert an object to sq*

Description

Takes an object of arbitrary type and returns an sq object as an output.

Usage

```
as.sq(x, ...)

## Default S3 method:
as.sq(x, ...)

## S3 method for class 'character'
as.sq(x, ...)
```

Arguments

x	[any] An object of a class that supports conversion to sq class.
...	further arguments to be passed from or to other methods.

Details

There are two possible cases: if x is a character vector, then this method calls sq function, else it passes x to import_sq and hopes it works.

Value

An sq object.

See Also

Functions from output module: as.character.sq(), as.matrix.sq(), export_sq(), write_fasta()

Examples

```
# Constructing an example sequence in the usual way:
sq_1 <- sq("CTGA")

# Using a method for character vector:
sq_2 <- as.sq("CTGA")

# Checking that both objects are identical:
identical(sq_1, sq_2)
```

---

bite *Subset sequences from sq objects*

Description

Extracts a defined range of elements from all sequences.

Usage

```
bite(x, indices, ...)

## S3 method for class 'sq'
bite(
  x,
  indices,
  ...,
  NA_letter = getOption("tidysq_NA_letter"),
  on_warning = getOption("tidysq_on_warning")
)
```

Arguments

x	[sq] An object this function is applied to.
indices	[integer] Indices to extract from each sequence. The function follows the normal R conventions for indexing vectors, including negative indices.
...	further arguments to be passed from or to other methods.
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".
on_warning	["silent" || "message" || "warning" || "error"] Determines the method of handling warning message. Default value is "warning".

Details

`bite` function allows user to access specific elements from multiple sequences at once.

By passing positive indices the user can choose, which elements they want from each sequence. If a sequence is shorter than an index, then `NA` value is inserted into the result in this place and a warning is issued. The user can specify behavior of R in this case by specifying `on_warning` parameter.

Negative indices are supported as well. Their interpretation is "to select all elements except those on positions specified by these negative indices". This means that e.g. `c(-1,-3,-5)` vector will be used to bite all sequence elements except the first, the third and the fifth. If a sequence is shorter than any index, then nothing happens, as it's physically impossible to extract an element at said index.

As per normal R convention, it isn't accepted to mix positive and negative indices, because there is no good interpretation possible for that.

Value

`sq` object of the same type as input `sq`, where each element is a subsequence created by indexing corresponding sequence from input `sq` object with input indices.

See Also

`remove_na`

Functions that affect order of elements: `collapse()`, `paste()`, `reverse()`

Examples

```
# Creating objects to work on:
sq_dna <- sq(c("ATGCAGGA", "GACCGNBAACGAN", "TGACGAGCTTA"),
             alphabet = "dna_bsc")
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_unt <- sq(c("ATGCAGGA?", "TGACGAGCTTA", "", "TIAALGNIIYRAIE"))

# Extracting first five letters:
bite(sq_dna, 1:5)

# If a sequence is shorter than 5, then NA is introduced:
bite(sq_unt, 1:5)

# Selecting fourth, seventh and fourth again letter:
bite(sq_ami, c(4, 7, 4))

# Selecting all letters except first four:
bite(sq_dna, -1:-4)
```

collapse *Collapse multiple sequences into one*

Description

Joins sequences from a vector into a single sequence. Sequence type remains unchanged.

Usage

```
collapse(x, ...)

## S3 method for class 'sq'
collapse(x, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

x	[sq] An object this function is applied to.
...	further arguments to be passed from or to other methods.
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

Details

collapse() joins sequences from supplied sq object in the same order as they appear in said vector. That is, if there are three sequences AGGCT, ATCCGT and GAACGT, then resulting sequence will be AGGCTATCCGTGAACGT. This operation does not alter the type of the input object nor its alphabet.

Value

sq object of the same type as input but with exactly one sequence.

See Also

Functions that affect order of elements: bite(), paste(), reverse()

Examples

```
# Creating objects to work on:
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_dna <- sq(c("ATGCAGGA", "GACCGAACGAN", ""), alphabet = "dna_ext")
sq_unt <- sq(c("ATGCAGGA?", "TGACGAGCTTA", "", "TIAALGNIIYRAIE"))

# Collapsing sequences:
collapse(sq_ami)
```

```
collapse(sq_dna)
collapse(sq_unt)

# Empty sq objects are collapsed as well (into empty string - ""):
sq_empty <- sq(character(), alphabet = "rna_bsc")
collapse(sq_empty)
```

`complement` *Create complement sequence from dnasq or rnasq object*

Description

Creates the complementary sequence from a given RNA or DNA sequence. The function keeps the type of sequence intact.

Usage

```
complement(x, ...)

## S3 method for class 'sq_dna_bsc'
complement(x, ..., NA_letter = getOption("tidysq_NA_letter"))

## S3 method for class 'sq_dna_ext'
complement(x, ..., NA_letter = getOption("tidysq_NA_letter"))

## S3 method for class 'sq_rna_bsc'
complement(x, ..., NA_letter = getOption("tidysq_NA_letter"))

## S3 method for class 'sq_rna_ext'
complement(x, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

`x`	[`sq_dna_bsc` || `sq_rna_bsc` || `sq_dna_ext` || `sq_rna_ext`] An object this function is applied to.
`...`	further arguments to be passed from or to other methods.
`NA_letter`	[`character(1)`] A string that is used to interpret and display `NA` value in the context of `sq class`. Default value equals to `"!"`.

Details

This function matches elements of sequence to their complementary letters. For unambiguous letters, `"C"` is matched with `"G"` and `"A"` is matched with either `"T"` (thymine) or `"U"` (uracil), depending on whether input is of **dna** or **rna** type.

Ambiguous letters are matched as well, for example `"N"` (any nucleotide) is matched with itself, while `"B"` (not alanine) is matched with `"V"` (not thymine/uracil).

Value

sq object of the same type as input but built of nucleotides complementary to those in the entered sequences.

See Also

sq

Functions interpreting sq in biological context: %has%(), find_motifs(), translate()

Examples

```
# Creating DNA and RNA sequences to work on:
sq_dna <- sq(c("ACTGCTG", "CTTAGA", "CCCT", "CTGAATGT"),
             alphabet = "dna_bsc")
sq_rna <- sq(c("BRAUDUG", "URKKBKUCA", "ANKRUGBNNG", "YYAUNAAAG"),
             alphabet = "rna_ext")

# Here complement() function is used to make PCR (Polymerase Chain Reaction)
# primers. Every sequence is rewritten to its complementary equivalent as
# in the following example: AAATTTGGG -> TTTAAACCC.

complement(sq_dna)
complement(sq_rna)

# Each sequence have now a complementary equivalent, which can be helpful
# during constructing PCR primers.
```

---

export_sq *Export sq objects into other formats*

Description

Converts object of class sq to a class from another package. Currently supported packages are **ape**, **bioseq**, **Bioconductor** and **seqinr**. For exact list of supported classes and resulting types, see details.

Usage

```
export_sq(x, export_format, name = NULL, ...)
```

Arguments

x	[sq] An object this function is applied to.
export_format	[character(1)] A string indicating desired class (with specified package for clarity).

name	[character] Vector of sequence names. Must be of the same length as sq object. Can be NULL.
...	further arguments to be passed from or to other methods.

Details

Currently supported formats are as follows (grouped by sq types):

- **ami**:
  - "ape::AAbin"
  - "bioseq::bioseq_aa"
  - "Biostrings::AAString"
  - "Biostrings::AAStringSet"
  - "seqinr::SeqFastaAA"
- **dna**:
  - "ape::DNAbin"
  - "bioseq::bioseq_dna"
  - "Biostrings::DNAString"
  - "Biostrings::DNAStringSet"
  - "seqinr::SeqFastadna"
- **rna**:
  - "bioseq::bioseq_rna"
  - "Biostrings::RNAString"
  - "Biostrings::RNAStringSet"

Value

An object with the format specified in the parameter. To find information about the detailed structure of this object, see documentation of these objects.

See Also

sq class

Functions from output module: as.character.sq(), as.matrix.sq(), as.sq(), write_fasta()

Examples

```
# DNA and amino acid sequences can be exported to most packages
sq_ami <- sq(c("MVVGL", "LAVPP"), alphabet = "ami_bsc")
export_sq(sq_ami, "ape::AAbin")
export_sq(sq_ami, "bioseq::bioseq_aa")
export_sq(sq_ami, "Biostrings::AAStringSet", c("one", "two"))
export_sq(sq_ami, "seqinr::SeqFastaAA")

sq_dna <- sq(c("TGATGAAGCGCA", "TTGATGGGAA"), alphabet = "dna_bsc")
export_sq(sq_dna, "ape::DNAbin", name = c("one", "two"))
```

```
export_sq(sq_dna, "bioseq::bioseq_dna")
export_sq(sq_dna, "Biostrings::DNAStringSet")
export_sq(sq_dna, "seqinr::SeqFastadna")

# RNA sequences are limited to Biostrings and bioseq
sq_rna <- sq(c("NUARYGCB", "", "DRKCNYBAU"), alphabet = "rna_ext")
export_sq(sq_rna, "bioseq::bioseq_rna")
export_sq(sq_rna, "Biostrings::RNAStringSet")

# Biostrings can export single sequences to simple strings as well
export_sq(sq_dna[1], "Biostrings::DNAString")
```

`find_invalid_letters` *Find elements which are not suitable for specified type.*

Description

Finds elements in given sequence not contained in amino acid or nucleotide alphabet.

Usage

```
find_invalid_letters(x, dest_type, ...)

## S3 method for class 'sq'
find_invalid_letters(
  x,
  dest_type,
  ...,
  NA_letter = getOption("tidysq_NA_letter")
)
```

Arguments

`x`	[sq] An object this function is applied to.
`dest_type`	[character(1)] The name of destination type - one of `"dna_bsc"`, `"dna_ext"`, `"rna_bsc"`, `"rna_ext"`, `"ami_bsc"` and `"ami_ext"`.
`...`	further arguments to be passed from or to other methods.
`NA_letter`	[character(1)] A string that is used to interpret and display `NA` value in the context of `sq class`. Default value equals to `"!"`.

Details

Amino acid, DNA and RNA standard alphabets have predefined letters. This function allows the user to check which letters from input sequences are not contained in selected one of these alphabets.

Returned list contains a character vector for each input sequence. Each element of a vector is a letter that appear in corresponding sequence and not in the target alphabet.

You can check which letters are valid for specified type in `alphabet` documentation.

Value

A list of mismatched elements for every sequence from `sq` object.

See Also

`alphabet()`

Functions that manipulate type of sequences: `is.sq()`, `sq_type()`, `substitute_letters()`, `typify()`

Examples

```
# Creating objects to work on:
sq_unt <- sq(c("ACGPOIUATTAGACG","GGATFGHA"), alphabet = "unt")
sq_ami <- sq(c("QWERTYUIZXCVBNM","LKJHGFDSAZXCVBN"), alphabet = "ami_ext")

# Mismatched elements might be from basic type:
find_invalid_letters(sq_ami, "ami_bsc")

# But also from type completely unrelated to the current one:
find_invalid_letters(sq_unt, "dna_ext")
```

---

find_motifs *Find given motifs*

Description

Finds all given motifs in sequences and returns their positions.

Usage

```
find_motifs(x, name, motifs, ...)

## S3 method for class 'sq'
find_motifs(x, name, motifs, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

x	[sq] An object this function is applied to.
name	[character] Vector of sequence names. Must be of the same length as sq object.
motifs	[character] Motifs to be searched for.
...	further arguments to be passed from or to other methods.
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

Details

This function allows search of a given motif or motifs in the sq object. It returns all motifs found with their start and end positions within a sequence.

Value

A tibble with following columns:

name	name of the sequence in which a motif was found
sought	sought motif
found	found subsequence, may differ from sought if the motif contained ambiguous letters
start	position of first element of found motif
end	position of last element of found motif

Motif capabilities and restrictions

There are more options than to simply create a motif that is a string representation of searched subsequence. For example, when using this function with any of standard types, i.e. **ami**, **dna** or **rna**, the user can create a motif with ambiguous letters. In this case the engine will try to match any of possible meanings of this letter. For example, take "B" from extended DNA alphabet. It means "not A", so it can be matched with "C", "G" and "T", but also "B", "Y" (either "C" or "T"), "K" (either "G" or "T") and "S" (either "C" or "G").

Full list of ambiguous letters with their meaning can be found on IUPAC site.

Motifs are also restricted in that the alphabets of sq objects on which search operations are conducted cannot contain "^" and "$" symbols. These two have a special meaning - they are used to indicate beginning and end of sequence respectively and can be used to limit the position of matched subsequences.

See Also

Functions interpreting sq in biological context: %has%(), complement(), translate()

Examples

```
# Creating objects to work on:
sq_dna <- sq(c("ATGCAGGA", "GACCGNBAACGAN", "TGACGAGCTTAG"),
             alphabet = "dna_bsc")
sq_ami <- sq(c("AGNTYIKFGGAYTI", "MATEGILIAADGYTWIL", "MIPADHICAANGIENAGIK"),
             alphabet = "ami_bsc")
sq_atp <- sq(c("mAmYmY", "nbAnsAmA", ""),
             alphabet = c("mA", "mY", "nbA", "nsA"))
sq_names <- c("sq1", "sq2", "sq3")

# Finding motif of two alanines followed by aspartic acid or asparagine
# ("AAB" motif matches "AAB", "AAD" and "AAN"):
find_motifs(sq_ami, sq_names, "AAB")

# Finding "C" at fourth position:
find_motifs(sq_dna, sq_names, "^NNNC")

# Finding motif "I" at second-to-last position:
find_motifs(sq_ami, sq_names, "IX$")

# Finding multiple motifs:
find_motifs(sq_dna, sq_names, c("^ABN", "ANCBY", "BAN$"))

# Finding multicharacter motifs:
find_motifs(sq_atp, sq_names, c("nsA", "mYmY$"))
```

---

get_sq_lengths *Get lengths of sequences in sq object*

Description

Returns number of elements in each sequence in given sq object.

Usage

```
get_sq_lengths(x)
```

Arguments

x	[sq] An object this function is applied to.

Details

Due to storage implementation, using lengths method returns length of stored raw vectors instead of real sequence lengths. This function accesses original_length attribute of each sequence, which attribute stores information about how many elements are there in given sequence.

Value

A numeric vector, where each element gives length of corresponding sequence from sq object.

See Also

Functions from utility module: ==.sq(), is.sq(), sqconcatenate, sqextract

Examples

```
# Creating objects to work on:
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_dna <- sq(c("ATGCAGGA", "GACCGAACGAN", "TGACGAGCTTA", "ACTNNAGCN"),
             alphabet = "dna_ext")

# Counting number of elements in sq object:
get_sq_lengths(sq_dna)
get_sq_lengths(sq_ami)
```

---

get_standard_alphabet *Get standard alphabet for given type.*

Description

Returns alphabet attribute of an object.

Usage

```
get_standard_alphabet(type)
```

Arguments

type	[character(1)] The name of standard sq type - one of "dna_bsc", "dna_ext", "rna_bsc", "rna_ext", "ami_bsc" and "ami_ext".

Details

Each of standard sq types has exactly one predefined alphabet. It allows **tidysq** to package to optimize type-specific operations like complement() or translate(). This function enables the user to access alphabet attribute common for all sq objects of given type.

For list of letters specific to any of these standard alphabets, see alphabet().

Value

An sq_alphabet object related to passed sq type.

See Also

Functions from alphabet module: `alphabet()`

`get_tidysq_options` *Obtain current state of tidysq options*

Description

Subsets all global options to display those related to **tidysq** package.

Usage

```
get_tidysq_options()
```

Details

The user can display value of selected option by calling `getOptions(option_name)` and set its value with `options(option_name = value)`, where `option_name` is an option name and `value` is a value to assign to an option.

Full list of options included in **tidysq** package is listed below:

- tidysq_NA_letter [`character(1)`]
  A letter to be used when printing, constructing or interpreting `NA` value. Defaults to `"!"`.
- tidysq_on_warning [`"silent" || "message" || "warning" || "error"`]
  Determines the method of handling warning message. Setting `"error"` makes any warning throw an exception and stop execution of the code. The difference between `"message"` and `"warning"` is that while both display warning text to the console, only the latter registers it so that it can be accessed with a call to `warnings()`. Lastly, `"silent"` setting causes any warnings to be completely ignored. Default value is `"warning"`.
- tidysq_pillar_max_width [codeinteger(1)]
  Determines max width of a column of `sq` class within a `tibble`. Default value is 15.
- tidysq_print_max_sequences [`integer(1)`]
  Controls maximum number of sequences printed to console. If an `sq` object is longer than this value, then only first `tidysq_print_max_sequences` are printed, just like in any R vector. Default value is 10.
- tidysq_print_use_color [`logical(1)`]
  Determines whether coloring should be used to increase readability of text printed to console. While it is advised to keep this option turned on due to above concern, some environments may not support coloring and thus turning it off can be necessary. Defaults to `TRUE`.
- tidysq_safe_mode [`logical(1)`]
  Default value is `FALSE`. When turned on, safe mode guarantees that `NA` appears within a sequence if and only if input sequence contains value passed with `NA_letter`. This means that resulting type might be different to the one passed as argument, if there are letters in a sequence that does not appear in the original alphabet.

Value

A named list with selected option values.

See Also

Functions that display sequence info: sqprint

`import_sq` *Import sq objects from other objects*

Description

Creates sq object from object of class from another package. Currently supported packages are **ape**, **bioseq**, **Bioconductor** and **seqinr**. For exact list of supported classes and resulting types, see details.

Usage

```
import_sq(object, ...)
```

Arguments

`object`	[any(1)] An object of one of supported classes.
`...`	further arguments to be passed from or to other methods.

Details

Currently supported classes are as follows:

- ape:
  - AAbin - imported as **ami_bsc**
  - DNAbin - imported as **dna_bsc**
  - alignment - exact type is guessed within sq function
- bioseq:
  - bioseq_aa - imported as **ami_ext**
  - bioseq_dna - imported as **dna_ext**
  - bioseq_rna - imported as **rna_ext**
- Biostrings:
  - AAString - imported as **ami_ext** with exactly one sequence
  - AAStringSet - imported as **ami_ext**
  - DNAString - imported as **dna_ext** with exactly one sequence
  - DNAStringSet - imported as **dna_ext**
  - RNAString - imported as **rna_ext** with exactly one sequence

  - `RNAStringSet` - imported as **rna_ext**
  - `BString` - imported as **unt** with exactly one sequence
  - `BStringSet` - imported as **unt**
  - `XStringSetList` - each element of a list can be imported as a separate `tibble`, resulting in a list of tibbles; if passed argument `separate = FALSE`, these tibbles are bound into one bigger tibble

- `seqinr`:
  - `SeqFastaAA` - imported as **ami_bsc**
  - `SeqFastadna` - imported as **dna_bsc**

Providing object of class other than specified will result in an error.

Value

A `tibble` with `sq` column of `sq` type representing the same sequences as given object; the object has a type corresponding to the input type; if given sequences have names, output `tibble` will also have another column `name` with those names

See Also

`sq class`

Functions from input module: `random_sq()`, `read_fasta()`, `sq()`

Examples

```
# ape example
library(ape)
ape_dna <- as.DNAbin(list(one = c("C", "T", "C", "A"), two = c("T", "G", "A", "G", "G")))
import_sq(ape_dna)

# bioseq example
library(bioseq)
bioseq_rna <- new_rna(c(one = "ANBRY", two = "YUTUGGN"))
import_sq(bioseq_rna)

# Biostrings example
library(Biostrings)
Biostrings_ami <- AAStringSet(c(one = "FEAPQLIWY", two = "EGITENAK"))
import_sq(Biostrings_ami)

# seqinr example
library(seqinr)
seqinr_dna <- as.SeqFastadna(c("C", "T", "C", "A"), name = "one")
import_sq(seqinr_dna)
```

`is.sq` *Check if object has specified type*

Description

Checks if object is an `sq` object without specifying type or if it is an `sq` object with specific type.

Usage

```
is.sq(x)

is.sq_dna_bsc(x)

is.sq_dna_ext(x)

is.sq_dna(x)

is.sq_rna_bsc(x)

is.sq_rna_ext(x)

is.sq_rna(x)

is.sq_ami_bsc(x)

is.sq_ami_ext(x)

is.sq_ami(x)

is.sq_unt(x)

is.sq_atp(x)
```

Arguments

`x`	[`sq`] An object this function is applied to.

Details

These functions are mostly simply calls to class checks. There are also grouped checks, i.e. `is.sq_dna`, `is.sq_rna` and `is.sq_ami`. These check for sq type regardless of if the type is basic or extended.

Value

A `logical` value - `TRUE` if `x` has specified type, `FALSE` otherwise.

See Also

Functions that manipulate type of sequences: find_invalid_letters(), sq_type(), substitute_letters(), typify()

Functions from utility module: ==.sq(), get_sq_lengths(), sqconcatenate, sqextract

Examples

```
# Creating objects to work on:
sq_dna <- sq(c("GGCAT", "TATC-A", "TGA"), alphabet = "dna_bsc")
sq_rna <- sq(c("CGAUUACG", "UUCUAGA", "UUCA"), alphabet = "rna_bsc")
sq_ami <- sq(c("CVMPQGQQ", "AHLC--PPQ"), alphabet = "ami_ext")
sq_unt <- sq("BAHHAJJ&HAN&JD&", alphabet = "unt")
sq_atp <- sq(c("mALPVQAmAmA", "mAmAPQ"), alphabet = c("mA", LETTERS))

# What is considered sq:
is.sq(sq_dna)
is.sq(sq_rna)
is.sq(sq_ami)
is.sq(sq_unt)
is.sq(sq_atp)

# What is not:
is.sq(c(1,2,3))
is.sq(LETTERS)
is.sq(TRUE)
is.sq(NULL)

# Checking for exact class:
is.sq_dna_bsc(sq_dna)
is.sq_dna_ext(sq_rna)
is.sq_rna_bsc(sq_ami)
is.sq_rna_ext(sq_rna)
is.sq_ami_bsc(sq_ami)
is.sq_ami_ext(sq_atp)
is.sq_atp(sq_atp)
is.sq_unt(sq_unt)

# Checking for generalized type:
is.sq_dna(sq_atp)
is.sq_rna(sq_rna)
is.sq_ami(sq_ami)
```

---

is_empty_sq *Test if sequence is empty*

Description

Test an sq object for presence of empty sequences.

Usage

```
is_empty_sq(x)
```

Arguments

x	[sq] An object this function is applied to.

Details

This function allows identification of empty sequences (that have length 0) represented by the NULL sq values in the sq object. It returns a logical value for every element of the sq object - TRUE if its value is NULL sq and FALSE otherwise. NULL sq values may be introduced as a result of remove_ambiguous and remove_na functions. The former replaces sequences containing ambiguous elements with NULL sq values, whereas the latter replaces sequences with NA values with NULL sq.

Value

A logical vector of the same length as input sq, indicating whether elements are empty sequences (of length 0).

See Also

sq class

Functions that clean sequences: remove_ambiguous(), remove_na()

Examples

```
# Creating an object to work on:
sq_dna_ext <- sq(c("ACGATTAGACG", "", "GACGANTCCAGNTAC"),
                 alphabet = "dna_ext")

# Testing for presence of empty sequences:
is_empty_sq(sq_dna_ext)

# Testing for presence of empty sequences after cleaning - sequence
# containing ambiguous elements is replaced by NULL sq:
sq_dna_bsc <- remove_ambiguous(sq_dna_ext)
is_empty_sq(sq_dna_bsc)

# Testing for presence of empty sequences after using bite and removing NA.
# Extracting letters from first to fifteenth - NA introduced:
bitten_sq <- bite(sq_dna_ext, 1:15)
# Removing NA:
rm_bitten_sq <- remove_na(bitten_sq)
# Testing for presence of empty sequences:
is_empty_sq(rm_bitten_sq)
```

paste *Paste sequences in string-like fashion*

Description

Joins multiple vectors of sequences into one vector.

Usage

```
## S3 method for class 'sq'
paste(..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

`...`	[sq] Sequences to paste together.
`NA_letter`	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

Details

paste() joins sequences in the same way as it does with strings. All sq objects must have the same length, that is, contain the same number of sequences. An exception is made for scalar (length 1) sq objects, which are replicated instead.

Value

sq object of common type of input objects. Common type is determined in the same process as for c.sq().

See Also

Functions that affect order of elements: bite(), collapse(), reverse()

Examples

```
# Creating objects to work on:
sq_dna_1 <- sq(c("TTCAGGGCTAG", "CGATTGC", "CAGTTTA"),
               alphabet = "dna_bsc")
sq_dna_2 <- sq(c("ATCTTGAAG", "CATATGCGCTA", "ACGTGTCGA"),
               alphabet = "dna_bsc")
sq_unt_1 <- sq(c("ATGCAGGA?", "TGACGAGCTTA", "", "TIAALGNIIYRAIE"))
sq_unt_2 <- sq(c("OVNU!!OK!!J", "GOK!MI!N!BB!", "DPOFIN!!", "??!?"))

# Pasting sequences:
collapse(sq_dna_1, sq_dna_2)
collapse(sq_unt_1, sq_unt_2)
```

```
collapse(sq_dna_2, sq_unt_2, sq_dna_1)
```

---

random_sq *Generate random sequences*

Description

Generates an sq object with specified number of sequences of given length and alphabet.

Usage

```
random_sq(n, len, alphabet, sd = NULL, use_gap = FALSE)
```

Arguments

n	[integer(1)] A number of sequences to generate - must be non-negative.
len	[integer(1)] Length of each sequence if sd not specified and mean length of sequences if sd specified - must be non-negative.
alphabet	[character] If provided value is a single string, it will be interpreted as type (see details). If provided value has length greater than one, it will be treated as atypical alphabet for sq object and sq type will be atp.
sd	[integer(1)] If specified, gives standard deviation of length of generated sequences - must be non-negative.
use_gap	[logical(1)] If TRUE, sequences will be generated with random gaps inside (commonly denoted as "-").

Details

Letter '*' is not used in generating **ami** sequences. If parameter sd is passed, then all generated negative values are replaced with 0s.

Value

An object of class sq with type as specified.

See Also

Functions from input module: import_sq(), read_fasta(), sq()

Examples

```
# Setting seed for reproducibility
set.seed(16)

# Generating random sequences
random_sq(10, 10, "ami_bsc")
random_sq(25, 18, "rna_bsc", sd = 6)
random_sq(50, 8, "dna_ext", sd = 3)
random_sq(6, 100, "ami_bsc", use_gap = TRUE)

# Passing whole alphabet instead of type
random_sq(4, 12, c("Pro", "Gly", "Ala", "Met", "Cys"))

# Generating empty sequences (why would anyone though)
random_sq(8, 0, "rna_ext")
```

---

read_fasta *Read a FASTA file*

Description

Reads a FASTA file that contains nucleotide or amino acid sequences and returns a tibble with obtained data.

Usage

```
read_fasta(
  file_name,
  alphabet = NULL,
  NA_letter = getOption("tidysq_NA_letter"),
  safe_mode = getOption("tidysq_safe_mode"),
  on_warning = getOption("tidysq_on_warning"),
  ignore_case = FALSE
)
```

Arguments

file_name	[character(1)] Absolute path to file or url to read from.
alphabet	[character] If provided value is a single string, it will be interpreted as type (see details). If provided value has length greater than one, it will be treated as atypical alphabet for sq object and sq type will be atp. If provided value is NULL, type guessing will be performed (see details).
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

safe_mode [logical(1)]
Default value is FALSE. When turned on, safe mode guarantees that NA appears within a sequence if and only if input sequence contains value passed with NA_letter. This means that resulting type might be different to the one passed as argument, if there are letters in a sequence that does not appear in the original alphabet.

on_warning ["silent" || "message" || "warning" || "error"]
Determines the method of handling warning message. Default value is "warning".

ignore_case [logical(1)]
If turned on, lowercase letters are turned into respective uppercase ones and interpreted as such. If not, either sq object must be of type **unt** or all lowercase letters are interpreted as NA values. Default value is FALSE. Ignoring case does not work with **atp** alphabets.

Details

All rules of creating sq objects are the same as in sq.

Value

A tibble with number of rows equal to the number of sequences and two columns:

- namespecifies name of a sequence, used in functions like find_motifs
- sqcontains extracted sequence itself

See Also

readLines

Functions from input module: import_sq(), random_sq(), sq()

Examples

```
fasta_file <- system.file(package = "tidysq", "examples/example_aa.fasta")

# In this case, these two calls are equivalent in result:
read_fasta(fasta_file)
read_fasta(fasta_file, alphabet = "ami_bsc")

## Not run:
# It's possible to read FASTA file from URL:
read_fasta("https://www.uniprot.org/uniprot/P28307.fasta")

## End(Not run)
```

`remove_ambiguous` *Remove sequences that contain ambiguous elements*

Description

This function replaces sequences with ambiguous elements by empty (`NULL`) sequences or removes ambiguous elements from sequences in an `sq` object.

Usage

```
remove_ambiguous(x, by_letter = FALSE, ...)

## S3 method for class 'sq'
remove_ambiguous(
  x,
  by_letter = FALSE,
  ...,
  NA_letter = getOption("tidysq_NA_letter")
)
```

Arguments

`x`	`[sq_dna_bsc || sq_rna_bsc || sq_dna_ext || sq_rna_ext || sq_ami_bsc || sq_ami_ext]` An object this function is applied to.
`by_letter`	`[logical(1)]` If `FALSE`, filter condition is applied to sequence as a whole. If `TRUE`, each letter is applied filter to separately.
`...`	further arguments to be passed from or to other methods.
`NA_letter`	`[character(1)]` A string that is used to interpret and display `NA` value in the context of `sq class`. Default value equals to `"!"`.

Details

Biological sequences, whether of DNA, RNA or amino acid elements, are not always exactly determined. Sometimes the only information the user has about an element is that it's one of given set of possible elements. In this case the element is described with one of special letters, here called **ambiguous**.

The inclusion of these letters is the difference between extended and basic alphabets (and, conversely, types). For amino acid alphabet these letters are: B, J, O, U, X, Z; whereas for DNA and RNA: W, S, M, K, R, Y, B, D, H, V, N.

`remove_ambiguous()` is used to create sequences without any of the elements above. Depending on value of `by_letter` argument, the function either replaces "ambiguous" sequences with empty sequences (if `by_letter` is equal to `TRUE`) or shortens original sequence by retaining only unambiguous letters (if opposite is true).

Value

An sq object with the **_bsc** version of inputted type.

See Also

Functions that clean sequences: is_empty_sq(), remove_na()

Examples

```
# Creating objects to work on:
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_dna <- sq(c("ATGCAGGA", "GACCGAACGAN", "TGACGAGCTTA", "ACTNNAGCN"),
             alphabet = "dna_ext")

# Removing whole sequences with ambiguous elements:
remove_ambiguous(sq_ami)
remove_ambiguous(sq_dna)

# Removing ambiguous elements from sequences:
remove_ambiguous(sq_ami, by_letter = TRUE)
remove_ambiguous(sq_dna, by_letter = TRUE)

# Analysis of the result
sq_clean <- remove_ambiguous(sq_ami)
is_empty_sq(sq_clean)
sq_type(sq_clean)
```

---

remove_na *Remove sequences that contain NA values*

Description

This function replaces sequences with NA values by empty (NULL) sequences or removes NA values from sequences in an sq object.

Usage

```
remove_na(x, by_letter = FALSE, ...)

## S3 method for class 'sq'
remove_na(x, by_letter = FALSE, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

x	[sq] An object this function is applied to.
by_letter	[logical(1)] If FALSE, filter condition is applied to sequence as a whole. If TRUE, each letter is applied filter to separately.
...	further arguments to be passed from or to other methods.
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

Details

NA may be introduced as a result of using functions like substitute_letters or bite. They can also appear in sequences if the user reads FASTA file using read_fasta or constructs sq object from character vector with sq function without safe_mode turned on - and there are letters in file or strings other than specified in the alphabet.

remove_na() is used to filter out sequences or elements that have NA value(s). By default, if any letter in a sequence is NA, then whole sequence is replaced by empty (NULL) sequence. However, if by_letter parameter is set to TRUE, then sequences are only shortened by excluding NA values.

Value

An sq object with the same type as the input type. Sequences that do not contain any NA values are left unchanged.

See Also

sq

Functions that clean sequences: is_empty_sq(), remove_ambiguous()

Examples

```
# Creating objects to work on:
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_dna <- sq(c("ATGCAGGA", "GACCGAACGAN", "TGACGAGCTTA", "ACTNNAGCN"),
             alphabet = "dna_ext")

# Substituting some letters with NA
sq_ami_sub <- substitute_letters(sq_ami, c(E = NA_character_, R = NA_character_))
sq_dna_sub <- substitute_letters(sq_dna, c(N = NA_character_))

# Biting sequences out of range
sq_bitten <- bite(sq_ami, 1:15)

# Printing the sequences
sq_ami_sub
sq_dna_sub
```

```
# Removing sequences containing NA
remove_na(sq_ami_sub)
remove_na(sq_dna_sub)
remove_na(sq_bitten)

# Removing only NA elements
remove_na(sq_ami_sub, by_letter = TRUE)
remove_na(sq_dna_sub, TRUE)
remove_na(sq_bitten, TRUE)
```

`reverse` *Reverse sequence*

Description

Reverse given list of sequences.

Usage

```
reverse(x, ...)

## S3 method for class 'sq'
reverse(x, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

`x`	[`sq`] An object this function is applied to.
`...`	further arguments to be passed from or to other methods.
`NA_letter`	[`character(1)`] A string that is used to interpret and display `NA` value in the context of `sq` `class`. Default value equals to `"!"`.

Details

`reverse()` function reverses each sequence in supplied `sq` object (e.q. transforms `"MIAANYTWIL"` to `"LIWTYNAAIM"`). This operation does not alter the type of the input object nor its alphabet.

Value

An `sq` object of the same type as input object but each sequence is reversed.

See Also

Functions that affect order of elements: `bite()`, `collapse()`, `paste()`

Examples

```
# Creating objects to work on:
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_dna <- sq(c("ATGCAGGA", "GACCGAACGAN", ""), alphabet = "dna_ext")
sq_unt <- sq(c("ATGCAGGA?", "TGACGAGCTTA", "", "TIAALGNIIYRAIE"))

# Reversing sequences:
reverse(sq_ami)
reverse(sq_dna)
reverse(sq_unt)
```

---

sq *Construct sq object from character vector*

Description

This function allows the user to construct objects of class sq from a character vector.

Usage

```
sq(
  x,
  alphabet = NULL,
  NA_letter = getOption("tidysq_NA_letter"),
  safe_mode = getOption("tidysq_safe_mode"),
  on_warning = getOption("tidysq_on_warning"),
  ignore_case = FALSE
)
```

Arguments

x	[character] Vector to construct sq object from.
alphabet	[character] If provided value is a single string, it will be interpreted as type (see details). If provided value has length greater than one, it will be treated as atypical alphabet for sq object and sq type will be atp. If provided value is NULL, type guessing will be performed (see details).
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".
safe_mode	[logical(1)] Default value is FALSE. When turned on, safe mode guarantees that NA appears within a sequence if and only if input sequence contains value passed with NA_letter. This means that resulting type might be different to the one passed

as argument, if there are letters in a sequence that does not appear in the original alphabet.

`on_warning` ["silent" || "message" || "warning" || "error"]
Determines the method of handling warning message. Default value is "warning".

`ignore_case` [logical(1)]
If turned on, lowercase letters are turned into respective uppercase ones and interpreted as such. If not, either sq object must be of type **unt** or all lowercase letters are interpreted as NA values. Default value is FALSE. Ignoring case does not work with **atp** alphabets.

Details

Function sq covers all possibilities of standard and non-standard types and alphabets. You can check what 'type' and 'alphabet' exactly are in sq class documentation. There is a guide below on how function operates and how the program behaves depending on arguments passed and letters in the sequences.

x parameter should be a character vector. Each element of this vector is a biological sequence. If this parameter has length 0, object of class sq with 0 sequences will be created (if not specified, it will have **dna_bsc** type, which is a result of rules written below). If it contains sequences of length 0, NULL sequences will be introduced (see *NULL (empty) sequences* section in sq class).

Important note: in all below cases word 'letter' stands for an element of an alphabet. Letter might consist of more than one character, for example "Ala" might be a single letter. However, if the user wants to construct or read sequences with multi-character letters, one has to specify all letters in alphabet parameter. Details of letters, alphabet and types can be found in sq class documentation.

Value

An object of class sq with appropriate type.

Simple guide to construct

In many cases, just the x parameter needs to be specified - type of sequences will be guessed according to rules described below. The user needs to pay attention, however, because for short sequences type may be guessed incorrectly - in this case they should specify type in alphabet parameter.

If your sequences contain non-standard letters, where each non-standard letter is one character long (that is, any character that is not an uppercase letter), you also don't need to specify any parameter. Optionally, you can explicitly do it by setting alphabet to "unt".

In safe mode it is guaranteed that only letters which are equal to NA_letter argument are interpreted as NA values. Due to that, resulting alphabet might be different from the alphabet argument.

Detailed guide to construct

Below are listed all possibilities that can occur during the construction of a sq object:

- If you don't specify any other parameter than x, function will try to guess sequence type (it will check in exactly this order):

1. If it contains only ACGT- letters, type will be set to **dna_bsc**.
2. If it contains only ACGU- letters, type will be set to **rna_bsc**.
3. If it contains any letters from 1. and 2. and additionally letters DEFHIKLMNPQRSVWY*, type will be set to **ami_bsc**.
4. If it contains any letters from 1. and additionally letters WSMKRYBDHVN, type will be set to **dna_ext**.
5. If it contains any letters from 2. and additionally letters WSMKRYBDHVN, type will be set to **rna_ext**.
6. If it contains any letters from previous points and additionally letters JOUXZ, type will be set to **ami_ext**.
7. If it contains any letters that exceed all groups mentioned above, type will be set to **unt**.

- If you specify `alphabet` parameter as any of `"dna_bsc"`, `"dna_ext"`, `"rna_bsc"`, `"rna_ext"`, `"ami_bsc"`, `"ami_ext"`; then:
  - If `safe_mode` is `FALSE`, then sequences will be built with standard alphabet for given type.
  - If `safe_mode` is `TRUE`, then sequences will be scanned for letters not in standard alphabet:
    - If no such letters are found, then sequences will be built with standard alphabet for given type.
    - If at least one such letter is found, then sequences are built with real alphabet and with type set to **unt**.
- If you specify `alphabet` parameter as `"unt"`, then sequences are scanned for alphabet and subsequently built with obtained alphabet and type **unt**.
- If you specify `alphabet` parameter as `character` vector longer than 1, then type is set to **atp** and alphabet is equal to letters in said parameter.

If `ignore_case` is set to `TRUE`, then lowercase letters are turned into uppercase during their interpretation, unless type is set to **atp**.

Handling unt and atp types and `NA` values

You can convert letters into another using `substitute_letters` and then use `typify` or `sq_type<-` function to set type of `sq` to **dna_bsc**, **dna_ext**, **rna_bsc**, **rna_ext**, **ami_bsc** or **ami_ext**. If your sequences contain `NA` values, use `remove_na`.

See Also

Functions from input module: `import_sq()`, `random_sq()`, `read_fasta()`

Examples

```
# constructing sq without specifying alphabet:
# Correct sq type will be guessed from appearing letters
## dna_bsc
sq(c("ATGC", "TCGTTA", "TT--AG"))

## rna_bsc
sq(c("CUUAC", "UACCGGC", "GCA-ACGU"))

## ami_bsc
```

```
sq(c("YQQPAVVM", "PQCFL"))

## ami cln sq can contain "*" - a letter meaning end of translation:
sq(c("MMDF*", "SYIHR*", "MGG*"))

## dna_ext
sq(c("TMVCCDA", "BASDT-CNN"))

## rna_ext
sq(c("WHDHKYN", "GCYVCYU"))

## ami_ext
sq(c("XYOQWWKCNJLO"))

## unt - assume that one wants to mark some special element in sequence with "%"
sq(c("%%YAPLAA", "PLAA"))

# passing type as alphabet parameter:
# All above examples yield an identical result if type specified is the same as guessed
sq(c("ATGC", "TCGTTA", "TT--AG"), "dna_bsc")
sq(c("CUUAC", "UACCGGC", "GCA-ACGU"), "rna_bsc")
sq(c("YQQPAVVM", "PQCFL"), "ami_bsc")
sq(c("MMDF*", "SYIHR*", "MGG*"), "ami_bsc")
sq(c("TMVCCDA", "BASDT-CNN"), "dna_ext")
sq(c("WHDHKYN", "GCYVCYU"), "rna_ext")
sq(c("XYOQWWKCNJLO"), "ami_ext")
sq(c("%%YAPLAA", "PLAA"), "unt")

# Type doesn't have to be the same as the guessed one if letters fit in the destination alphabet
sq(c("ATGC", "TCGTTA", "TT--AG"), "dna_ext")
sq(c("ATGC", "TCGTTA", "TT--AG"), "ami_bsc")
sq(c("ATGC", "TCGTTA", "TT--AG"), "ami_ext")
sq(c("ATGC", "TCGTTA", "TT--AG"), "unt")

# constructing sq with specified letters of alphabet:
# In sequences below "mA" denotes methyled alanine - two characters are treated as single letter
sq(c("LmAQYmASSR", "LmASMKLKFmAmA"), alphabet = c("mA", LETTERS))
# Order of alphabet letters are not meaningful in most cases
sq(c("LmAQYmASSR", "LmASMKLKFmAmA"), alphabet = c(LETTERS, "mA"))

# reading sequences with three-letter names:
sq(c("ProProGlyAlaMetAlaCys"), alphabet = c("Pro", "Gly", "Ala", "Met", "Cys"))

# using safe mode:
# Safe mode guarantees that no element is read as NA
# But resulting alphabet might be different to the passed one (albeit with warning/error)
sq(c("CUUAC", "UACCGGC", "GCA-ACGU"), alphabet = "dna_bsc", safe_mode = TRUE)
sq(c("CUUAC", "UACCGGC", "GCA-ACGU"), alphabet = "dna_bsc")

# Safe mode guesses alphabet based on whole sequence
long_sequence <- paste0(paste0(rep("A", 4500), collapse = ""), "N")
sq(long_sequence, safe_mode = TRUE)
sq(long_sequence)
```

```
# ignoring case:
# By default, lower- and uppercase letters are treated separately
# This behavior can be changed by setting ignore_case = TRUE
sq(c("aTGc", "tcgTTA", "tt--AG"), ignore_case = TRUE)
sq(c("XYOqwwKCNJLo"), ignore_case = TRUE)

# It is possible to construct sq with length 0
sq(character())

# As well as sq with empty sequences
sq(c("AGTGGC", "", "CATGA", ""))
```

`sq-class` *sq: class for keeping biological sequences tidy*

Description

An object of class **sq** represents a list of biological sequences. It is the main internal format of the **tidysq** package and most functions operate on it. The storage method is memory-optimized so that objects require as little memory as possible (details below).

Construction/reading/import of sq objects

There are multiple ways of obtaining `sq` objects:

- constructing from a `character` vector with `sq` method,
- constructing from another object with `as.sq` method,
- reading from the FASTA file with `read_fasta`,
- importing from a format of other package like **ape** or **Biostrings** with `import_sq`.

Important note: A manual assignment of a class `sq` to an object is **strongly discouraged** - due to the usage of low-level functions for bit packing such assignment may lead to calling one of those functions during operating on object or even printing it which can cause a crash of R session and, in consequence, loss of data.

Export/writing of sq objects

There are multiple ways of saving `sq` objects or converting them into other formats:

- converting into a character vector with `as.character` method,
- converting into a character matrix with `as.matrix` method,
- saving as FASTA file with `write_fasta`,
- exporting into a format of other package like `ape` or `Biostrings` with `export_sq`.

Ambiguous letters

This package is meant to handle amino acid, DNA and RNA sequences. IUPAC standard for one letter codes includes ambiguous bases that are used to describe more than one basic standard base. For example, "B" in the context of DNA code means "any of C, G or T". As there are operations that make sense only for unambiguous bases (like `translate`), this package has separate types for sequences with "basic" and "extended" alphabet.

Types of sq

There is need to differentiate `sq` objects that keep different types of sequences (DNA, RNA, amino acid), as they use different alphabets. Furthermore, there are special types for handling non-standard sequence formats.

Each **sq** object has exactly one of **types**:

- **ami_bsc** - (*amino acids*) represents a list of sequences of amino acids (peptides or proteins),
- **ami_ext** - same as above, but with possible usage of ambiguous letters,
- **dna_bsc** - (*DNA*) represents a list of DNA sequences,
- **dna_ext** - same as above, but with possible usage of ambiguous letters,
- **rna_bsc** - (*RNA*) represents a list of RNA sequences (together with DNA above often collectively called "nucleotide sequences"),
- **rna_ext** - same as above, but with possible usage of ambiguous letters,
- **unt** - (*untyped*) represents a list of sequences that do not have specified type. They are mainly result of reading sequences from a file that contains some letters that are not in standard nucleotide or amino acid alphabets and user has not specified them explicitly. They should be converted to other **sq** classes (using functions like `substitute_letters` or `typify`),
- **atp** - (*atypical*) represents sequences that have an alphabet different from standard alphabets - similarly to **unt**, but user has been explicitly informed about it. They are result of constructing sequences or reading from file with provided custom alphabet (for details see `read_fasta` and `sq` function). They are also result of using function `substitute_letters` - users can use it to for example simplify an alphabet and replace several letters by one.

For clarity, **ami_bsc** and **ami_ext** types are often referred to collectively as **ami** when there is no need to explicitly specify every possible type. The same applies to **dna** and **rna**.

`sq` object type is printed when using overloaded method `print`. It can be also checked and obtained as a value (that may be passed as argument to function) by using `sq_type`.

Alphabet

See `alphabet`.

The user can obtain an alphabet of the `sq` object using the `alphabet` function. The user can check which letters are invalid (i.e. not represented in standard amino acid or nucleotide alphabet) in each sequence of given `sq` object by using `find_invalid_letters`. To substitute one letter with another use `substitute_letters`.

Missing/Not Available values

There is a possibility of introducing NA values into sequences. NA value does not represents gap (which are represented by "-") or wildcard elements ("N" in the case of nucleotides and "X" in the case of amino acids), but is used as a representation of an empty position or invalid letters (not represented in nucleotide or amino acid alphabet).

NA does not belong to any alphabet. It is printed as "!" and, thus, it is highly unrecommended to use "!" as special letter in **atp** sequences (but print character can be changed in options, see tidysq-options).

NA might be introduced by:

- reading fasta file with non-standard letters with read_fasta with safe_mode argument set to TRUE,
- replacing a letter with NA value with substitute_letters,
- subsetting sequences beyond their lengths with bite.

The user can convert sequences that contain NA values into NULL sequences with remove_na.

NULL (empty) sequences

NULL sequence is a sequence of length 0.

NULL sequences might be introduced by:

- constructing sq object from character string of length zero,
- using the remove_ambiguous function,
- using the remove_na function,
- subsetting sq object with bite function (and negative indices that span at least -1:-length(sequence).

Storage format

sq object is, in fact, **list of raw vectors**. The fact that it is list implies that the user can concatenate sq objects using c method and subset them using extract operator. Alphabet is kept as an attribute of the object.

Raw vectors are the most efficient way of storage - each letter of a sequence is assigned an integer (its index in alphabet of sq object). Those integers in binary format fit in less than 8 bits, but normally are stored on 16 bits. However, thanks to bit packing it is possible to remove unused bits and store numbers more tightly. This means that all operations must either be implemented with this packing in mind or accept a little time overhead induced by unpacking and repacking sequences. However, this cost is relatively low in comparison to amount of saved memory.

For example - **dna_bsc** alphabet consists of 5 values: ACGT-. They are assigned numbers 0 to 4 respectively. Those numbers in binary format take form: 000, 001, 010, 011, 100. Each of these letters can be coded with just 3 bits instead of 8 which is demanded by char - this allows us to save more than 60% of memory spent on storage of basic nucleotide sequences.

tibble compatibility

sq objects are compatible with tibble class - that means one can have an sq object as a column of a tibble. There are overloaded print methods, so that it is printed in pretty format.

`sqapply` *Apply function to each sequence*

Description

Applies given function to each sequence. Sequences are passed to function as character vectors (or numeric, if type of `sq` is **enc**) or single character strings, depending on parameter.

Usage

```
sqapply(
  x,
  fun,
  ...,
  single_string = FALSE,
  NA_letter = getOption("tidysq_NA_letter")
)
```

Arguments

`x`	[`sq`] An object this function is applied to.
`fun`	[`function(1)`] A function to apply to each sequence in `sq` object; it should take a character vector, numeric vector or single character string as an input.
`...`	further arguments to be passed from or to other methods.
`single_string`	[`logical(1)`] A value indicating in which form sequences should be passed to the function `fun`; if `FALSE` (default), they will be treated as character vectors, if `TRUE`, they will be pasted into a single string.
`NA_letter`	[`character(1)`] A string that is used to interpret and display `NA` value in the context of `sq class`. Default value equals to "`!`".

Value

A list of values returned by function for each sequence in corresponding order.

See Also

`sq lapply`

Examples

```
# Creating objects to work on:
sq_dna <- sq(c("ATGCAGGA", "GACCGNBAACGAN", "TGACGAGCTTA"),
             alphabet = "dna_bsc")
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_unt <- sq(c("ATGCAGGA?", "TGACGAGCTTA", "", "TIAALGNIIYRAIE"))

# Counting how may "A" elements are present in sequences:

sqapply(sq_dna, function(sequence) sum(sequence == "A"))
sqapply(sq_ami, function(sequence) sum(sequence == "A"))
sqapply(sq_unt, function(sequence) sum(sequence == "A"))
```

`sqconcatenate` *Concatenate sq objects*

Description

Merges multiple sq and possibly `character` objects into one larger `sq` object.

Arguments

`...`	[`sq` || `character`] Multiple objects. For exact behavior, check Details section. First argument must be of `sq` class due to R mechanism of single dispatch. If this is a problem, recommended alternative is `vec_c` method from `vctrs-package` package.

Details

Whenever all passed objects are of one of standard types (that is, **dna_bsc**, **dna_ext**, **rna_bsc**, **rna_ext**, **ami_bsc** or **ami_ext**), returned object is of the same class, as no changes to alphabet are needed.

It's possible to mix both basic and extended types within one call to `c()`, however they all must be of the same type (that is, either **dna**, **rna** or **ami**). In this case, returned object is of extended type.

Mixing **dna**, **rna** and **ami** types is prohibited, as interpretation of letters differ depending on the type.

Whenever all objects are either of **atp** type, returned object is also of this class and resulting alphabet is equal to set union of all input alphabets.

unt type can be mixed with any other type, resulting in **unt** object with alphabet equal to set union of all input alphabets. In this case, it is possible to concatenate **dna** and **ami** objects, for instance, by concatenating one of them first with **unt** object. However, it is strongly discouraged, as it may result in unwanted concatenation of DNA and amino acid sequences.

Whenever a character vector appears, it does not influence resulting sq type. Each element is treated as separate sequence. If any of letters in this vector does not appear in resulting alphabet, it is silently replaced with `NA`.

Due to R dispatch mechanism passing character vector as first will return class-less list. This behavior is effectively impossible and definitely unrecommended to fix, as fixing it would involve changing c primitive. If such possibility is necessary, vec_c is a better alternative.

Value

sq object with length equal to sum of lengths of individual objects passed as parameters. Elements of sq are concatenated just as if they were normal lists (see c).

See Also

Functions from utility module: ==.sq(), get_sq_lengths(), is.sq(), sqextract

Examples

```
# Creating objects to work on:
sq_dna_1 <- sq(c("GGACTGCA", "CTAGTA", ""), alphabet = "dna_bsc")
sq_dna_2 <- sq(c("ATGACA", "AC-G", "-CCAT"), alphabet = "dna_bsc")
sq_dna_3 <- sq(character(), alphabet = "dna_bsc")
sq_dna_4 <- sq(c("BNACV", "GDBADHH"), alphabet = "dna_ext")
sq_rna_1 <- sq(c("UAUGCA", "UAGCCG"), alphabet = "rna_bsc")
sq_rna_2 <- sq(c("-AHVRYA", "G-U-HYR"), alphabet = "rna_ext")
sq_rna_3 <- sq("AUHUCHYRBNN--", alphabet = "rna_ext")
sq_ami <- sq("ACHNK-IFK-VYW", alphabet = "ami_bsc")
sq_unt <- sq("AF:gf;PPQ^&XN")

# Concatenating dna_bsc sequences:
c(sq_dna_1, sq_dna_2, sq_dna_3)
# Concatenating rna_ext sequences:
c(sq_rna_2, sq_rna_3)
# Mixing dna_bsc and dna_ext:
c(sq_dna_1, sq_dna_4, sq_dna_2)

# Mixing DNA and RNA sequences doesn't work:
## Not run:
c(sq_dna_3, sq_rna_1)

## End(Not run)

# untsq can be mixed with DNA, RNA and amino acids:
c(sq_ami, sq_unt)
c(sq_unt, sq_rna_1, sq_rna_2)
c(sq_dna_2, sq_unt, sq_dna_3)

# Character vectors are also acceptable:
c(sq_dna_2, "TGCA-GA")
c(sq_rna_2, c("UACUGGGACUG", "AUGUBNAABNRYYRAU"), sq_rna_3)
c(sq_unt, "&#JIA$O02t30,9ec", sq_ami)
```

`sqextract` *Extract parts of a sq object*

Description

Operator to extract subsets of sq objects.

Arguments

`x`	[`sq`] An object this function is applied to.
`i, j, ...`	[`numeric` || `logical`] Indices specifying elements to extract.

Details

This function follows `vctrs-package` conventions regarding argument interpretation for indexing vectors, which are a bit stricter that normal R conventions, for example implicit argument recycling is prohibited. Subsetting of the `sq` object does not affect its attributes (class and alphabet of the object). Attempt to extract elements using indices not present in the object will return an error.

Value

`sq` object of the same type as the input, containing extracted elements

See Also

Functions from utility module: `==.sq()`, `get_sq_lengths()`, `is.sq()`, `sqconcatenate`

Examples

```
# Creating object to work on:
sq_unt <- sq(c("AHSNLVSCTK$SH%&VS", "YQTVKA&#BSKJGY",
               "IAKVGDCTWCTY&GT", "AVYI#VSV&*DVGDJCFA"))

# Subsetting using numeric vectors
# Extracting second element of the object:
sq_unt[2]

# Extracting elements from second to fourth:
sq_unt[2:4]

# Extracting all elements except the third:
sq_unt[-3]

# Extracting first and third element:
sq_unt[c(1,3)]

# Subsetting using logical vectors
```

```
# Extracing first and third element:
sq_unt[c(TRUE, FALSE, TRUE, FALSE)]

# Subsetting using empty vector returns all values:
sq_unt[]

# Using NULL, on the other hand, returns empty sq:
sq_unt[NULL]
```

`sqprint` *Print sq object*

Description

Prints input `sq` object in a human-friendly form.

Arguments

`x`	[`sq`] An object this function is applied to.
`max_sequences`	[`integer(1)`] How many sequences should be printed.
`use_color`	[`logical(1)`] Should sequences be colored?
`letters_sep`	[`character(1)`] How the letters should be separated.
`NA_letter`	[`character(1)`] A string that is used to interpret and display `NA` value in the context of `sq` `class`. Default value equals to "`!`".
`...`	further arguments to be passed from or to other methods.

Details

`print` method is often called implicitly by calling variable name. Only explicit calling of this method allows its parameters to be changed.

Printed information consists of three parts:

- First line is always a header that contains info about the type of sequences contained.
- The next part is the content. Each sequence has its own line, but not all sequences are printed. The number of printed sequences is limited by parameter `max_sequences`, defaulting to 10. These sequences are printed with:
  - left-aligned index of sequence in square brackets (e.g. `[3]`),
  - left-aligned sequence data (more about it in paragraph below),
  - right-aligned sequence length in angle brackets (e.g. `<27>`).

- Finally, if number of sequences is greater than `max_sequences`, then a footer is displayed with how many sequences are there and how many were printed.

Each sequence data is printed as letters. If sequence is too long to fit in one line, then only a subsequence is displayed - a subsequence that begins from the first letter. Sequence printing is controlled by `letters_sep` and `NA_letter` parameters. The first one specifies a string that should be inserted between any two letters. By default it's empty when all letters are one character in length; and a space otherwise. `NA_letter` dictates how `NA` values are displayed, by default it's an exclamation mark (`"!"`).

Most consoles support color printing, but when any of these do not, then the user might use `use_color` parameter set to `FALSE` - or better yet, change related option value, where said option is called `"tidysq_print_use_color"`.

Value

An object that was passed as the first argument to the function. It is returned invisibly (equivalent of `invisible(x)`)

See Also

Functions that display sequence info: `get_tidysq_options()`

Examples

```
# Creating objects to work on:
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_dna <- sq(c("ATGCAGGA", "GACCGNBAACGAN", "TGACGAGCTTA"),
             alphabet = "dna_bsc")
sq_unt <- sq(c("ATGCAGGA?", "TGACGAGCTTA", "", "TIAALGNIIYRAIE"))

# Printing without explicit function calling with default parameters:
sq_ami
sq_dna
sq_unt

# Printing with explicit function calling and specific parameters:
print(sq_ami)
print(sq_dna, max_sequences = 1, use_color = FALSE)
print(sq_unt, letters_sep = ":")
```

`sq_type` *Get type of an sq object*

Description

Returns type of sequences/alphabet contained in `sq` object.

Usage

```
sq_type(x, ...)

## S3 method for class 'sq'
sq_type(x, ...)

sq_type(x) <- value

## S3 replacement method for class 'sq'
sq_type(x) <- value
```

Arguments

x	[sq] An object this function is applied to.
...	further arguments to be passed from or to other methods.
value	[character(1)] The name of destination type - any valid sq type.

Details

Types returned by this function can be passed as argument to functions like random_sq and find_invalid_letters.

Value

A string, one of: "ami_bsc", "ami_ext", "dna_bsc", "dna_ext", "rna_bsc", "rna_ext", "unt" or "atp".

See Also

sq class

Functions that manipulate type of sequences: find_invalid_letters(), is.sq(), substitute_letters(), typify()

Examples

```
# Creating objects to work on:
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_dna <- sq(c("ATGCAGGA", "GACCGAACGA", "TGACGAGCTTA", "ACTTTAGC"),
             alphabet = "dna_bsc")

# Extracting type of sq objects:
sq_type(sq_ami)
sq_type(sq_dna)

# Classes are tightly related to these types:
class(sq_ami)[1]
class(sq_dna)[1]
```

`substitute_letters` *Substitute letters in a sequence*

Description

Replaces all occurrences of a letter with another.

Usage

```
substitute_letters(x, encoding, ...)

## S3 method for class 'sq'
substitute_letters(x, encoding, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

`x`	[`sq`] An object this function is applied to.
`encoding`	[`character` || `numeric`] A dictionary (named vector), where names are letters to be replaced and elements are their respective replacements.
`...`	further arguments to be passed from or to other methods.
`NA_letter`	[`character(1)`] A string that is used to interpret and display `NA` value in the context of `sq` `class`. Default value equals to `"!"`.

Details

`substitute_letters` allows to replace unwanted letters in any sequence with user-defined or IUPAC symbols. Letters can also be replaced with `NA` values, so that they can be later removed from the sequence by `remove_na` function.

It doesn't matter whether replaced or replacing letter is single or multiple character. However, the user cannot replace multiple letters with one nor one letter with more than one.

Of course, multiple different letters can be encoded to the same symbol, so `c(A = "rep1", H = "rep1", G = "rep1")` is allowed, but `c(AHG = "rep1")` is not (unless there is a letter `"AHG"` in the alphabet). By doing that any information of separateness of original letters is lost, so it isn't possible to retrieve original sequence after this operation.

All encoding names must be letters contained within the alphabet, otherwise an error will be thrown.

Value

An `sq` object of **atp** type with updated alphabet.

See Also

Functions that manipulate type of sequences: `find_invalid_letters()`, `is.sq()`, `sq_type()`, `typify()`

Examples

```
# Creating objects to work on:
sq_dna <- sq(c("ATGCAGGA", "GACCGAACGAN", "TGACGAGCTTA", "ACTNNAGCN"),
             alphabet = "dna_ext")
sq_ami <- sq(c("MIOONYTWIL","TIOOLGNIIYROIE", "NYERTGHLI", "MOYXXXIOLN"),
             alphabet = "ami_ext")
sq_atp <- sq(c("mALPVQAmAmA", "mAmAPQ"), alphabet = c("mA", LETTERS))

# Not all letters must have their encoding specified:
substitute_letters(sq_dna, c(T = "t", A = "a", C = "c", G = "g"))
substitute_letters(sq_ami, c(M = "X"))

# Multiple character letters are supported in encodings:
substitute_letters(sq_atp, c(mA = "-"))
substitute_letters(sq_ami, c(I = "ough", O = "eau"))

# Numeric substitutions are allowed too, these are coerced to characters:
substitute_letters(sq_dna, c(N = 9, G = 7))

# It's possible to replace a letter with NA value:
substitute_letters(sq_ami, c(X = NA_character_))
```

---

translate *Convert DNA or RNA into proteins using genetic code*

Description

This function allows the user to input DNA or RNA sequences and acquire sequences of corresponding proteins, where correspondence is encoded in specified table.

Usage

```
translate(x, table = 1, ...)

## S3 method for class 'sq_dna_bsc'
translate(x, table = 1, ..., NA_letter = getOption("tidysq_NA_letter"))

## S3 method for class 'sq_rna_bsc'
translate(x, table = 1, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

x	[sq_dna_bsc || sq_rna_bsc] An object this function is applied to.
table	[integer(1)] The number of translation table used, as specified here.
...	further arguments to be passed from or to other methods.

NA_letter [character(1)]
A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

Details

DNA and RNA sequences use combinations of three consecutive nucleic acids to encode one of 22 amino acids. This encoding is called "genetic code".

translate() first splits passed DNA or RNA sequences into three-letter chunks. Then searches the codon table for the entry where the key is equal to the current chunk and the value is one letter that encodes the corresponding protein. These resulting letters are then pasted into one sequence for each input sequence.

Due to how the tables works, translate() does not support inputting sequences with extended alphabets, as ambiguous letters in most cases cannot be translated into exactly one protein.

Moreover, this function raises an error whenever input sequence contain either "-" or NA value.

Value

An object of class sq with **ami_bsc** type.

See Also

remove_ambiguous, substitute_letters and typify for necessary actions before using translate()

Functions interpreting sq in biological context: %has%(), complement(), find_motifs()

Examples

```
sq_dna <- sq(c("TACTGGGCATGA", "CAGGTC", "TAGTCCTAG"), alphabet = "dna_bsc")
translate(sq_dna)
```

---

typify *Set type of an sq object*

Description

Sets sequence type (and, consequently, alphabet attribute) to one of **ami**, **dna** or **rna** types.

Usage

```
typify(x, dest_type, ...)

## S3 method for class 'sq'
typify(x, dest_type, ..., NA_letter = getOption("tidysq_NA_letter"))
```

Arguments

x	[sq] An object this function is applied to.
dest_type	[character(1)] The name of destination type - one of "dna_bsc", "dna_ext", "rna_bsc", "rna_ext", "ami_bsc" and "ami_ext".
...	further arguments to be passed from or to other methods.
NA_letter	[character(1)] A string that is used to interpret and display NA value in the context of sq class. Default value equals to "!".

Details

Sometimes functions from I/O module return sequences of incorrect type, most often **unt** (which indicates no type). It happens mostly whenever there are letters that don't fit into target alphabet. After replacing wrong letters with correct ones with substitute_letters the user has sequences of type **atp**, even if their alphabet is contained in the target one. At the same time, many functions demand sequences to be of standard type (i.e. **ami**, **dna** or **rna**) or behave differently for these.

typify() is used to help with these situations by allowing the user to convert their sequences to target type. There are some conditions that must be met to use this function. The most important is that typified sq object must not contain invalid letters. If this condition is not satisfied, an error is thrown.

If dest_type is equal to type of sq, function simply returns input value.

Value

sq object with the same letters as input x, but with type as specified in dest_type.

See Also

Functions that manipulate type of sequences: find_invalid_letters(), is.sq(), sq_type(), substitute_letters()

Examples

```
# Constructing sq object with strange characters (type will be set to "unt"):
sq_unt <- sq(c("&VPLG&#", "##LCG"))

# Substituting letters with "X", which stands for unknown amino acid:
sq_sub <- substitute_letters(sq_unt, c(`&` = "X", `#` = "X"))

# Setting extended amino acid type (only extended one has "X" letter):
typify(sq_sub, "ami_ext")
```

`write_fasta` *Save sq to fasta file*

Description

Writes `sq` objects with their names to a fasta file.

Usage

```
write_fasta(
  x,
  name,
  file,
  width = 80,
  NA_letter = getOption("tidysq_NA_letter")
)
```

Arguments

`x`	[`sq`] An object this function is applied to.
`name`	[`character`] Vector of sequence names. Must be of the same length as `sq` object.
`file`	[`character(1)`] Absolute path to file to write to.
`width`	[`integer(1)`] Maximum number of characters to put in each line of file. Must be positive.
`NA_letter`	[`character(1)`] A string that is used to interpret and display `NA` value in the context of `sq` `class`. Default value equals to "`!`".

Details

Whenever a name has more letters than `width` parameter, nothing happens, as only sequences are split to fit within designated space.

Value

No value is returned.

See Also

Functions from output module: `as.character.sq()`, `as.matrix.sq()`, `as.sq()`, `export_sq()`

Examples

```
## Not run:
sq_dna <- sq(c("ACTGCTG", "CTTAGA", "CCCT", "CTGAATGT"),
             alphabet = "dna_bsc")
write_fasta(sq_dna,
            c("bat", "cat", "rat", "elephant_swallowed_by_A_snake"),
            "~/fasta_rubbish/example.fasta")

## End(Not run)
```

%has% *Test sq object for presence of given motifs*

Description

Tests if elements of a sq object contain given motifs.

Usage

```
x %has% y
```

Arguments

x	[sq] An object this function is applied to.
y	[character] Motifs to be searched for.

Details

This function allows testing if elements of a sq object contain the given motif or motifs. It returns a logical value for every element of the sq object - TRUE if tested sequence contains searched motif and FALSE otherwise. When multiple motifs are searched, TRUE will be returned only for sequences that contain all given motifs.

This function only indicates if a motif is present within a sequence, to find all motifs and their positions within sequences use find_motifs.

Value

A logical vector of the same length as input sq, indicating which elements contain all given motifs.

Motif capabilities and restrictions

There are more options than to simply create a motif that is a string representation of searched subsequence. For example, when using this function with any of standard types, i.e. **ami**, **dna** or **rna**, the user can create a motif with ambiguous letters. In this case the engine will try to match any of possible meanings of this letter. For example, take "B" from extended DNA alphabet. It means "not A", so it can be matched with "C", "G" and "T", but also "B", "Y" (either "C" or "T"), "K" (either "G" or "T") and "S" (either "C" or "G").

Full list of ambiguous letters with their meaning can be found on IUPAC site.

Motifs are also restricted in that the alphabets of `sq` objects on which search operations are conducted cannot contain "^" and "$" symbols. These two have a special meaning - they are used to indicate beginning and end of sequence respectively and can be used to limit the position of matched subsequences.

See Also

Functions interpreting sq in biological context: `complement()`, `find_motifs()`, `translate()`

Examples

```
# Creating objects to work on:
sq_dna <- sq(c("ATGCAGGA", "GACCGNBAACGAN", "TGACGAGCTTAG"),
             alphabet = "dna_bsc")
sq_ami <- sq(c("MIAANYTWIL","TIAALGNIIYRAIE", "NYERTGHLI", "MAYXXXIALN"),
             alphabet = "ami_ext")
sq_atp <- sq(c("mAmYmY", "nbAnsAmA", ""),
             alphabet = c("mA", "mY", "nbA", "nsA"))

# Testing if DNA sequences contain motif "ATG":
sq_dna %has% "ATG"

# Testing if DNA sequences begin with "ATG":
sq_dna %has% "^ATG"

# Testing if DNA sequences end with "TAG" (one of the stop codons):
sq_dna %has% "TAG$"

# Test if amino acid sequences contain motif of two alanines followed by
# aspartic acid or asparagine ("AAB" motif matches "AAB", "AAD" and "AAN"):
sq_ami %has% "AAB"

# Test if amino acid sequences contain both motifs:
sq_ami %has% c("AAXG", "MAT")

# Test for sequences with multicharacter alphabet:
sq_atp %has% c("nsA", "mYmY$")
```

Index