Package 'transmogR'

November 12, 2026

Type Package

Title Modify a set of reference sequences using a set of variants

Version 1.8.1

Description transmogR provides the tools needed to crate a new reference genome or reference transcriptome, using a set of variants. Variants can be any combination of SNPs, Insertions and Deletions. The intended use-case is to enable creation of variant-modified reference transcriptomes for incorporation into transcriptomic pseudo-alignment workflows, such as salmon.

License GPL-3

Encoding UTF-8

URL https://github.com/smped/transmogR

BugReports https://github.com/smped/transmogR/issues

Depends R (>= 4.1.0), Biostrings, GenomicRanges

Imports BSgenome, data.table, Seqinfo, GenomicFeatures, ggplot2 (>= 4.0.0), IRanges, jsonlite, matrixStats, methods, parallel, patchwork, scales, stats, S4Vectors, SummarizedExperiment, VariantAnnotation

Suggests BiocStyle, BSgenome.Hsapiens.UCSC.hg38, edgeR, extraChIPs, InteractionSet, knitr, readr, rmarkdown, rtracklayer, SimpleUpset, testthat (>= 3.0.0)

biocViews Alignment, GenomicVariation, Sequencing, TranscriptomeVariant, VariantAnnotation

BiocType Software

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

git_url https://git.bioconductor.org/packages/transmogR

git_branch RELEASE_3_23

git_last_commit 6ae2ef3

git_last_commit_date 2026-06-30

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Stevie Pederson [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8197-3303>)

Maintainer Stevie Pederson <stephen.pederson.au@gmail.com>

Contents

transmogR-package 2
cleanVariants 3
digestSalmon 5
genomogrify 7
indelcator 9
overlapsByVar 11
owl 12
parY 13
shiftByVar 14
sjFromExons 15
transmogrify 17
upsetVarByCol 20
varTags 22
varTypes 23

Index **24**

`transmogR-package` *transmogR: Create a variant-modified reference transcriptome*

Description

The package `transmogR` has been designed for creation of a variant-modified reference transcriptome

Details

The package `transmogR` provides two primary functions for modifying complete transcriptomes or genomes:

- `transmogrify()` for incorporating the supplied variants into transcriptomic sequences, and
- `genomogrify()` for incorporating the supplied variants into genomic sequences, ideally to be passed as decoy sequences to a tool such as `salmon`.

The main functions rely on lower-level functions such as:

- `owl()` which over-writes letters (i.e. SNPs) within a sequence, and
- `indelcator()` which incorporates InDels into an individual sequence

Additional utility functions are provided which allow characterisation and exploration of any set of variants:

- `overlapsByVar()` counts the variants which overlap sets of GenomicRanges, first splitting the variants into SNV, Insertions and Deletions
- `parY()` returns the pseudo-autosomal regions for a chosen genome build as a GenomicRanges object
- `upsetVarByCol()` produces an UpSet plot counting how many unique IDs are impacted by a set o variants. IDs can represent any column in the supplied ranges, such as gene_id or transcript_id
- `varTypes()` classifies a set of variants into SNV, Insertions of Deletions

Author(s)

Stevie Pederson

See Also

Useful links:

- https://github.com/smped/transmogR
- Report bugs at https://github.com/smped/transmogR/issues

`cleanVariants` *Check provided variants for compatibility*

Description

Check and cleanup a set of variants for downstream compatibility

Usage

```
cleanVariants(var, ...)

## S4 method for signature 'GRanges'
cleanVariants(var, ol_vars = "fail", ref_col = "REF", alt_col = "ALT", ...)

## S4 method for signature 'VcfFile'
cleanVariants(
  var,
  ol_vars = "fail",
  which,
  ref_col = "REF",
  alt_col = "ALT",
  ...
)
```

Arguments

var	GRanges object containing the variants, or a VariantAnnotation::VcfFile
...	Not used
ol_vars	Error handling for any overlapping variants. Can take values in c("fail", "none", "first", "last", "longest", "shortest"). Default is set to fail, with additional options to drop all overlapping variants ('none'), select by genomic position ('first', 'last'), or select by the scale of change to the genome ('longest', 'shortest')
ref_col, alt_col	Column names corresponding to the reference and alternate alleles
which	Passed to VariantAnnotation::ScanVcfParam if working with a VcfFile

Details

This function checks a set of variants for the expected structure which is required by all downstream functions in transmogR. The primary change to the data structure is that both REF and ALT columns will be set as simple character vectors.

Given the complicated variant calls that can often be produced by variant callers, additional checks performed will be to ensure that:

- there are no overlapping variants
- SNPs are all single nucleotides and not longer strings
- SNPs are bi-allelic
- Insertions contain a single nucleotide in the REF column
- Deletions contain a single nucleotide in the ALT column
- No missing values are present
- All ALT/REF nucleotides conform to IUPAC codings

Value

GRanges object with any incompatible variants removed, or an error produced. The mcols will contain the columns REF and ALT, unless otherwise specified, as character vectors

Examples

```
# Any conflicting variants will be removed
var <- GRanges(c("chr10:114468420-114468422", "chr10:114468422"))
var$REF <- c("GCC", "C")
var$ALT <- c("G", "CTAT")
var
## Taken from the 1000GP, the first variant would delete the C at 114468422
## whilst the second variant begins an insertion at this position.
## These are clearly conflicting. The default value for ol_vars is to fail
## with an error (ol_vars = "fail"). However, both can be removed by setting
## ol_vars = "none". A warning will always be produced.
cleanVariants(var, ol_vars = "none")
## Or the longest can be retained, along with multiple other options
cleanVariants(var, ol_vars = "longest")
```

digestSalmon *Parse the output from salmon*

Description

Parse transcript counts and additional data from salmon

Calculate the overdispersions from a set of paths without parsing any counts

Usage

```
digestSalmon(
  paths,
  max_sets = 2L,
  aux_dir = "aux_info",
  name_fun = basename,
  verbose = TRUE,
  extra_assays = NULL,
  max_boot = Inf,
  ...
)

overdispFromBoots(paths, n_boot, .ids)
```

Arguments

paths	Vector of file paths to directories containing salmon results
max_sets	The maximum number of indexes permitted
aux_dir	Subdirectory where bootstraps and meta_info.json are stored
name_fun	Function applied to paths to provide colnames in the returned object. Set to NULL or c() to disable.
verbose	Print progress messages
extra_assays	Can take values in c("TPM", "effectiveLength", "length") to optionally request TPM, effectiveLength or length as assays. Including the length assay is intended for the use case of personalised transcriptomes where transcript lengths may no longer be uniform across samples. None will be returned by default
max_boot	The maximum number of bootstraps to use. Setting this to zero will ignore all bootstraps and the scaledCounts assay will not be included in the returned object
...	Not used
n_boot	The number of bootstraps
.ids	Vector of transcript IDs which match the bootstrap values. Will be parsed from paths if not provided, although this adds time

Details

This function is based heavily on `edgeR::catchSalmon()` however, there are some important differences:

1. A SummarizedExperiment object is returned
2. Differing numbers of transcripts are allowed between samples

The second point is intended for the scenario where some samples may have been aligned to a full reference, with remaining samples aligned to a partially masked reference (e.g. chrY). This will lead to differing numbers of transcripts within each salmon index, however, common estimates of overdispersions are required for scaling transcript-level counts. By default, the function will error if >2 different sets of transcripts are detected, however this can be modified using the max_sets argument.

This greater flexibility also requires more stringent checking and, as such, for smaller datasets, digestSalmon may be slower that the edgeR function.

The SummarizedExperiment object returned may also contain multiple assays, as described elsewhere on this page

This follows the methods of Baldoni, et al. (2024). Dividing out quantification uncertainty allows efficient assessment of differential transcript expression with edgeR. Nucleic Acids Research, 52(3), e13. https://doi.org/10.1093/nar/gkad1167

Value

A SummarizedExperiment object containing assays for counts and scaledCounts. The scaledCounts assay contains counts divided by overdispersions. rowData in the returned object will also include transcript-lengths along with the overdispersion estimates used to return the scaled counts. TPM, effectiveLength and length can be returned as additional assays by specifying one or more of these in the extra_assays argument

`overdispFromBoots` returns a numeric vector

Examples

```
## Provide the path to the parent directories which contains each
## quant.sf file
ex_path <- system.file("extdata/salmon_test", package = "transmogR")
se <- digestSalmon(ex_path, extra_assays = "TPM", verbose = FALSE)
se

ex_path <- system.file("extdata/salmon_test", package = "transmogR")
overdispFromBoots(ex_path, 10)
```

genomogrify *Mogrify a genome using a set of variants*

Description

Use a set of SNPS, insertions and deletions to modify a reference genome

Usage

```
genomogrify(x, var, ...)

## S4 method for signature 'XStringSet,GRanges'
genomogrify(
  x,
  var,
  alt_col = "ALT",
  mask = GRanges(),
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  ...
)

## S4 method for signature 'BSgenome,GRanges'
genomogrify(
  x,
  var,
  alt_col = "ALT",
  mask = GRanges(),
  names,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  ...
)

## S4 method for signature 'BSgenome,VcfFile'
genomogrify(
  x,
  var,
  alt_col = "ALT",
```

```
  mask = GRanges(),
  names,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  which,
  verbose = TRUE,
  ...
)

## S4 method for signature 'XStringSet,VcfFile'
genomogrify(
  x,
  var,
  alt_col = "ALT",
  mask = GRanges(),
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  which,
  verbose = TRUE,
  ...
)
```

Arguments

x	A DNAStringSet or BSgenome
var	GRanges object containing the variants, or a VariantAnnotation::VcfFile
...	Passed to parallel::mclapply
alt_col	The name of the column with var containing alternate bases
mask	Optional GRanges object defining regions to be masked with an 'N'
tag	Optional tag to add to all sequence names which were modified
sep	Separator to place between seqnames names & tag
var_tags	logical(1) Add tags indicating which type of variant were incorporated, with 's', 'i' and 'd' representing SNPs, Insertions and Deletions respectively
var_sep	Separator between any previous tags and variant tags
ol_vars	Error handling for any overlapping variants. See cleanVariants for possible values and an explanation
verbose	logical(1) Print progress messages while running
names	Sequence names to be mogrified
which	GRanges object passed to VariantAnnotation::ScanVcfParam if using a VCF directly

Details

This function is designed to create a variant-modified reference genome, intended to be included as a set of decoys when using salmon in selective alignment mode. Sequence lengths will change if InDels are included and any coordinate-based information will be lost on the output of this function.

Tags are able to be added to any modified sequence to assist identifying any changes that have been made to a sequence.

Value

XStringSet with variant modified sequences

Examples

```
library(GenomicRanges)
dna <- DNAStringSet(c(chr1 = "ACGT", chr2 = "AATTT"))
var <- GRanges(c("chr1:1", "chr1:3", "chr2:1-3"))
var$ALT <- c("C", "GG", "A")
dna
genomogrify(dna, var)
genomogrify(dna, var, tag = "mod")
genomogrify(dna, var, var_tags = TRUE)
genomogrify(dna, var, mask = GRanges("chr2:1-5"), var_tags = TRUE)
```

---

indelcator *Substitute InDels into one or more sequences*

Description

Modify one or more sequences to include Insertions or Deletions

Usage

```
indelcator(x, indels, ...)

## S4 method for signature 'XString,GRanges'
indelcator(x, indels, exons, alt_col = "ALT", ol_vars = "fail", ...)

## S4 method for signature 'DNAStringSet,GRanges'
indelcator(
  x,
  indels,
  alt_col = "ALT",
  ol_vars = "fail",
  mc.cores = 1,
  verbose = TRUE,
```

```
  ...
)

## S4 method for signature 'BSgenome,GRanges'
indelcator(
  x,
  indels,
  alt_col = "ALT",
  ol_vars = "fail",
  mc.cores = 1,
  names,
  ...
)
```

Arguments

x	Sequence of class XString
indels	GRanges object with InDel locations and the alternate allele
...	Passed to parallel::mclapply
exons	GRanges object containing exon structure for x
alt_col	Column containing the alternate allele
ol_vars	Error handling for any overlapping variants. See cleanVariants for possible values and an explanation
mc.cores	Number of cores to use when calling parallel::mclapply internally
verbose	logical(1) Print all messages
names	passed to BSgenome::getSeq() when x is a BSgenome object

Details

This is a lower-level function relied on by both `transmogrify()` and `genomogrify()`.

Takes an Biostrings::XString or Biostrings::XStringSet object and modifies the sequence to incorporate InDels. The expected types of data determine the behaviour, with the following expectations describing how the function will incorporate data

Input Data Type	Exons Required	Use Case	Returned
XString	Y	Modify a Reference Transcriptome	XString
DNAStringSet	N	Modify a Reference Genome	DNAStringSet
BSgenome	N	Modify a Reference Genome	DNAStringSet

Value

A DNAStringSet or XString object (See Details)

See Also

`transmogrify()` `genomogrify()`

Examples

```
## Start with a DNAStringSet
library(GenomicRanges)
seq <- DNAStringSet(c(seq1 = "AATCTGCGC"))
## Define an Insertion
var <- GRanges("seq1:1")
var$ALT <- "AAA"
seq
indelcator(seq, var)

## To modify a single transcript
library(GenomicFeatures)
ex <- GRanges(c("seq1:1-3:+", "seq1:7-9:+"))
orig <- extractTranscriptSeqs(seq, GRangesList(tx1 = ex))[["tx1"]]
orig
indelcator(orig, var, exons = ex)
```

`overlapsByVar` *Count overlaps by variant type*

Description

Count how many variants of each type overlap ranges

Usage

```
overlapsByVar(x, var, ...)

## S4 method for signature 'GRangesList,GRanges'
overlapsByVar(x, var, alt_col = "ALT", ...)

## S4 method for signature 'GRanges,GRanges'
overlapsByVar(x, var, alt_col = "ALT", ...)
```

Arguments

`x`	A GRangesList with features of interest
`var`	A Granges object with variants of interest
`...`	Passed to rowSums
`alt_col`	The column within mcols(var) which contains the alternate allele

Details

Taking any GRanges or GRangesList, count how many of each variant type overlap a region.

Value

A vector or matrix

Examples

```
library(rtracklayer)
library(VariantAnnotation)
gtf <- import.gff(
    system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR")
)
grl <- splitAsList(gtf, gtf$type)
vcf <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
var <- rowRanges(readVcf(vcf, param = ScanVcfParam(fixed = "ALT")))
overlapsByVar(grl, var)
```

---

owl *OverWrite Letters in an XStringSet*

Description

OverWrite Letters (e.g. SNPs) in an XStringSet

Usage

```
owl(seq, snps, ...)

## S4 method for signature 'XStringSet,GRanges'
owl(seq, snps, alt_col = "ALT", ol_vars = "fail", ...)

## S4 method for signature 'BSgenome,GRanges'
owl(seq, snps, alt_col = "ALT", names, ...)
```

Arguments

seq	A BSgenome, DNAStringSet, RNAStringSet or other XStringSet.
snps	A GRanges object with SNP positions and a column containing the alternate allele
...	Passed to Biostrings::replaceLetterAt()
alt_col	Column name in the mcols element of snps containing the alternate allele
ol_vars	Error handling for any overlapping variants. See cleanVariants for possible values and an explanation
names	Sequence names to operate on

Details

This is a lower-level function called by `transmogrify()` and `genomogrify()`, but able to be called by the user if needed

Note that when providing a BSgenome object, this will first be coerced to a DNAStringSet which can be time consuming.

Value

An object of the same class as the original object, but with SNPs inserted at the supplied positions

Examples

```
seq <- DNAStringSet(c(chr1 = "AAGC"))
snps <- GRanges("chr1:2")
snps$ALT <- "G"
snps
seq
owl(seq, snps)
```

---

parY *Get the PAR-Y Regions From a Seqinfo Object*

Description

Define the Pseudo-Autosomal Regions from a Seqinfo Object

Usage

```
parY(x, ...)

## S4 method for signature 'Seqinfo'
parY(x, ...)

## S4 method for signature 'character'
parY(x, prefix = NULL, ...)
```

Arguments

`x`	A Seqinfo object or any of named build. If passing a character vector, `match.arg()` will be used to match the build.
`...`	Not used
`prefix`	Optional prefix to place before chromosome names. Can only be NULL, "" or "chr"

Details

Using a seqinfo object based on either hg38, hg19, CHM13.v2 or their variations, create a GRanges object with the Pseudo-Autosomal Regions from the Y chromosome for that build. The length of the Y chromosome on the seqinfo object is used to determine the correct genome build when passing a Seqinfo object. Otherwise

An additional mcols column called PAR will indicate PAR1 and PAR2

Value

A GenomicRanges object

Examples

```
library(Seqinfo)
sq <- Seqinfo(
    seqnames = "chrY", seqlengths = 59373566, genome = "hg19_only_chrY"
)
parY(sq)

## PAR regions for CHM13 are also available
sq <- Seqinfo(
    seqnames = "chrY", seqlengths = 62460029, genome = "CHM13"
)
parY(sq)

## Or just call by name
parY("GRCh38", prefix = "chr")
```

---

shiftByVar *Calculate new exon co-ordinates*

Description

Calcluate new exon co-ordinates after including InDels

Usage

```
shiftByVar(x, var, alt_col = "ALT", mc.cores = 1, ...)
```

Arguments

x	A GenomicRanges object with co-ordinates needing to be recalculated
var	A set of variants to be incorporated into a reference genome
alt_col	The name of the column with the alternate sequence for each variant
mc.cores	Passed internally to `parallel::mclapply()`
...	Not used

Details

Given a set of variants, this will return a set of genomic ranges with updated co-ordinates able to be applied on a variant modified reference genome

Value

GRanges object with co-ordinates shifted according the to the provided variants. The new co-ordinates will be compatible with a variant-modified genome as produced by `genomogrify()` and can be used to extract the sequences associated with the ranges in the modified reference.

Examples

```
# Define a 3nt insertion
var <- GRanges("seq1:5:*", seqlengths = c(seq1=10), REF = "A", ALT = "AGT")
var
# A simple GRanges to shift co-ordinates for
gr <- GRanges("seq1:1-10:+", seqlengths = c(seq1=10), feature = "feature1")
gr
# Create shifted co-ordinates based on the provided variants
new_gr <- shiftByVar(gr, var)
new_gr
## The seqlengths will have been adjusted to account for all variants
seqinfo(new_gr)
```

---

sjFromExons *Obtain Splice-Junctions from Exons and Transcripts*

Description

Using GRanges defining exons and transcripts, find the splice-junctions

Usage

```
sjFromExons(
  x,
  rank_col = c("exon_number", "exon_rank"),
  tx_col = c("transcript_id", "tx_id"),
  extra_cols = "all",
  don_len = 8,
  acc_len = 5,
  as = c("GRanges", "GInteractions"),
  ...
)
```

Arguments

x	GRanges object with exons and transcripts. A column indicating the position (or rank) of each exon within the transcript must be included.
rank_col	The column containing the position of each exons within the transcript
tx_col	The column containing unique transcript-level identifiers
extra_cols	Can be a vector of column names to return beyond rank_col and tx_col. By default all columns are returned (extra_cols = "all").
don_len, acc_len	Length of donor and acceptor sites respectively
as	Return as a set of GenomicRanges, or with each splice junction annotated as a GenomicInteraction
...	Not used

Details

A canonical splice junction consists of a donor site and an acceptor site at each end of an intron, with a branching site somewhere wthin the intron. Canonical donor sites are 8nt long with the the first two bases being exonic and the next 6 being derived form intronic sequences. Canonical acceptor sites are 5nt long with the first four bases being intronic and the final base being the first base of the next exon.

This functions uses each set of exons within a transcript to identify both donor and acceptor sites. Branch sites are not identified.

Value

A GRanges object with requested columns, and an additional column, 'site', annotating each region as a donor or acceptor site.

Alternatively, by specifying as = "GInteractions", the junctions can be returned with each splice junction annotated as a GenomicInteraction. This can make the set of junctions easier to interpret for a given transcript.

Examples

```
library(rtracklayer)
gtf_cols <- c(
  "transcript_id", "transcript_name", "gene_id", "gene_name", "exon_number"
)
gtf <- import.gff(
   system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR"),
   feature.type = "exon", colnames = gtf_cols
)
sj <- sjFromExons(gtf)
sj

## Or to simplify shared splice junctions across multiple transcripts
library(extraChIPs, quietly = TRUE)
chopMC(sj)
```

```
## Splice Junctions can also be returned as a GInteractions object with
## anchorOne as the donor & anchorTwo as the acceptor sites
sjFromExons(gtf, as = "GInteractions")
```

---

transmogrify *Mogrify a transcriptome using a set of variants*

Description

Use a set of SNPs, insertions and deletions to modify a reference transcriptome

Usage

```
transmogrify(x, var, exons, ...)

## S4 method for signature 'XStringSet,GRanges,GRanges'
transmogrify(
  x,
  var,
  exons,
  alt_col = "ALT",
  trans_col = "transcript_id",
  omit_ranges = NULL,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  mc.cores = 1,
  ...
)

## S4 method for signature 'BSgenome,GRanges,GRanges'
transmogrify(
  x,
  var,
  exons,
  alt_col = "ALT",
  trans_col = "transcript_id",
  omit_ranges = NULL,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
```

```
  ol_vars = "fail",
  verbose = TRUE,
  mc.cores = 1,
  ...
)

## S4 method for signature 'BSgenome,VcfFile,GRanges'
transmogrify(
  x,
  var,
  exons,
  alt_col = "ALT",
  trans_col = "transcript_id",
  omit_ranges = NULL,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  mc.cores = 1,
  which,
  ...
)

## S4 method for signature 'XStringSet,VcfFile,GRanges'
transmogrify(
  x,
  var,
  exons,
  alt_col = "ALT",
  trans_col = "transcript_id",
  omit_ranges = NULL,
  tag = NULL,
  sep = "_",
  var_tags = FALSE,
  var_sep = "_",
  ol_vars = "fail",
  verbose = TRUE,
  mc.cores = 1,
  which,
  ...
)
```

Arguments

x	Reference genome as either a DNAStringSet or BSgenome
var	GRanges object containing the variants

exons	GRanges object with ranges representing exons
...	Passed to parallel::mclapply
alt_col	Column from var containing alternate bases
trans_col	Column from 'exons' containing the transcript_id
omit_ranges	GRanges object containing ranges to omit, such as PAR-Y regions, for example
tag	Optional tag to add to all sequence names which were modified
sep	Separator to place between seqnames names & tag
var_tags	logical(1) Add tags indicating which type of variant were incorporated, with 's', 'i' and 'd' representing SNPs, Insertions and Deletions respectively
var_sep	Separator between any previous tags and variant tags
ol_vars	Error handling for any overlapping variants. See cleanVariants for possible values and an explanation
verbose	logical(1) Include informative messages, or operate silently
mc.cores	Number of cores to be used when multi-threading via parallel::mclapply
which	GRanges object passed to VariantAnnotation::ScanVcfParam if using a VCF directly

Details

Produce a set of variant modified transcript sequences from a standard reference genome. Supported variants are SNPs, Insertions and Deletions

Ranges needing to be masked, such as the Y-chromosome, or Y-PAR can be provided.

It should be noted that this is a time consuming process Inclusion of a large set of insertions and deletions across an entire transcriptome can involve individually modifying many thousands of transcripts, which can be a computationally demanding task. Whilst this can be parallelised using an appropriate number of cores, this may also prove taxing for lower power laptops, and pre-emptively closing memory hungry programs such as Slack, or internet browers may be prudent.

Value

An XStringSet

Examples

```
library(GenomicRanges)
library(GenomicFeatures)
seq <- DNAStringSet(c(chr1 = "ACGTAAATGG"))
exons <- GRanges(c("chr1:1-3:-", "chr1:7-9:-"))
exons$transcript_id <- c("trans1")

# When using extractTranscriptSeqs -stranded exons need to be sorted by end
exons <- sort(exons, decreasing = TRUE, by = ~end)
exons
trByExon <- splitAsList(exons, exons$transcript_id)

# Check the sequences
```

```
seq
extractTranscriptSeqs(seq, trByExon)

# Define some variants
var <- GRanges(c("chr1:2", "chr1:8"))
var$ALT <- c("A", "GGG")

# Include the variants adding tags to indicate a SNP and indel
# The exons GRanges object will be split by transcript internally
transmogrify(seq, var, exons, var_tags = TRUE)
```

`upsetVarByCol` *Show Variants by Impacted Columns*

Description

Produce an UpSet plot showing unique values from a given column

Usage

```
upsetVarByCol(
  gr,
  var,
  alt_col = "ALT",
  mcol = "transcript_id",
  ...,
  fill = NULL,
  fill_scale = scale_fill_discrete(),
  expand_sets = 0.2,
  hj_sets = 1.1,
  expand_intersect = 0.1,
  vj_intersect = -0.5,
  label_size = 3.5,
  title
)
```

Arguments

`gr`	GRanges object with ranges representing a key feature such as exons
`var`	GRanges object with variants in a given column
`alt_col`	Column within `var` containing the alternate allele
`mcol`	The column within `gr` to summarise results by
`...`	Passed to SimpleUpset::simpleUpSet
`fill`	Optional column in gr used to fill intersections and sets

fill_scale	Discrete ggplot2 scale for filling bars. Ignored if fill = NULL
expand_sets	Expand the set-size x-axis by this amount
hj_sets	Horizontal adjustment of set size labels
expand_intersect	Expand the intersection y-axis by this amount
vj_intersect	Vertical adjustment of intersection size labels
label_size	Control the size of both intersection and sit size labels
title	Summary title to show above the intersection panel. Can be hidden by setting to NULL

Details

Take a set of variants, classify them as SNV, Insertion and Deletion, then using a GRanges object, produce an UpSet plot showing impacted values from a given column

Value

An UpSet plot

See Also

SimpleUpset::simpleUpSet

Examples

```
library(rtracklayer)
library(VariantAnnotation)
library(ggplot2)
gtf <- import.gff(
    system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR"),
    feature.type = "exon"
)
vcf <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
var <- rowRanges(readVcf(vcf, param = ScanVcfParam(fixed = "ALT")))
upsetVarByCol(gtf, var)
upsetVarByCol(
  gtf, var, fill = "transcript_type",
  fill_scale = scale_fill_brewer(palette = "Set1")
)
```

varTags *Create a set of tags indicating overlap status with variants*

Description

Create a set of tags indicating overlap status with variants

Usage

```
varTags(x, var, tag = NULL, var_tags = TRUE, sep = "_", pre = sep, ...)
```

Arguments

x	GRanges or GRangesList
var	Set of variants for x to be compared to
tag	Tag to be added for all overlapping positions
var_tags	logical(1) Include 's', 'i' and 'd' tags. See details
sep	Separator added between tag and var_tags
pre	Separator to add at the start of returned tags
...	Passed to cleanVariants()

Details

Take a GRanges or GRangesList and compare against a set of variants. Variants will be classified into SNV, Insertions and Deletions using varTypes() and tags defined. An overall set of tags defining any overlap can be created by themselves. An additional set of tags containing 's', 'i' or 'd' to indicate overlap with an SNV, Insertion or Deletion can also be created, with the concatentation of both tags being returned.

Value

Character vector of the same length as x

Examples

```
# Load the included subset of 1000 Genomes Variants
library(VariantAnnotation)
vcf <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
vcf <- VcfFile(vcf)
var <- cleanVariants(vcf)
# Now load some exons, then split by transcript, subsetting to the first 40
library(rtracklayer)
f <- system.file("extdata/gencode.v44.subset.gtf.gz", package = "transmogR")
gtf <- import.gff(f, feature.type = "exon")
exon_by_trans <- splitAsList(gtf, gtf$transcript_id)[1:40]
# And produce tags based on the overlapping variants within the exons
# Overlapping SNVs will return an 's' whilst insertions include an 'i'
```

```
varTags(exon_by_trans, var, tag = "1000GP")
```

---

varTypes *Identify SNVs, Insertions and Deletions*

Description

Identify SNVs, Insertions and Deletions within a GRanges object

Usage

```
varTypes(x, alt_col = "ALT", ...)
```

Arguments

x	GenomicRanges object
alt_col	Name of the column with mcols(x) which contains the alternate allele. Can be an XStringSetList, XStringSet or character
...	Not used

Details

Using the width of the reference and alternate alleles, classify each range as an SNV, Insertion or Deletion.

- SNVs are expected to have REF & ALT widths of 1
- Insertions are expected to have ALT longer than REF
- Deletions are expected to have ALT shorter than REF

These are relatively permissive criteria

Value

Character vector

Examples

```
# Load the example VCF and classify ranges
library(VariantAnnotation)
f <- system.file("extdata/1000GP_subset.vcf.gz", package = "transmogR")
vcf <- readVcf(f)
gr <- rowRanges(vcf)
type <- varTypes(gr)
table(type)
gr[type != "SNV"]
```

Index

∗ **internal**
    transmogR-package, 2

Biostrings::replaceLetterAt(), *12*
Biostrings::XString, *10*
Biostrings::XStringSet, *10*

cleanVariants, 3, *8*, *10*, *12*, *19*
cleanVariants(), *22*
cleanVariants,GRanges-method *(*cleanVariants*)*, 3
cleanVariants,VcfFile-method *(*cleanVariants*)*, 3
cleanVariants-methods *(*cleanVariants*)*, 3

digestSalmon, 5

edgeR::catchSalmon(), *6*

genomogrify, 7
genomogrify(), *2*, *10*, *13*, *15*
genomogrify,BSgenome,GRanges-method *(*genomogrify*)*, 7
genomogrify,BSgenome,VcfFile-method *(*genomogrify*)*, 7
genomogrify,XStringSet,GRanges-method *(*genomogrify*)*, 7
genomogrify,XStringSet,VcfFile-method *(*genomogrify*)*, 7
genomogrify-methods *(*genomogrify*)*, 7

indelcator, 9
indelcator(), *3*
indelcator,BSgenome,GRanges-method *(*indelcator*)*, 9
indelcator,DNAStringSet,GRanges-method *(*indelcator*)*, 9
indelcator,XString,GRanges-method *(*indelcator*)*, 9

match.arg(), *13*

overdispFromBoots *(*digestSalmon*)*, 5
overlapsByVar, 11
overlapsByVar(), *3*
overlapsByVar,GRanges,GRanges-method *(*overlapsByVar*)*, 11
overlapsByVar,GRangesList,GRanges-method *(*overlapsByVar*)*, 11
overlapsByVar-methods *(*overlapsByVar*)*, 11
owl, 12
owl(), *3*
owl,BSgenome,GRanges-method *(*owl*)*, 12
owl,XStringSet,GRanges-method *(*owl*)*, 12

parallel::mclapply, *8*, *10*, *19*
parallel::mclapply(), *14*
parY, 13
parY(), *3*
parY,character-method *(*parY*)*, 13
parY,Seqinfo-method *(*parY*)*, 13
parY-methods *(*parY*)*, 13

rowSums, *11*

shiftByVar, 14
SimpleUpset::simpleUpSet, *20*, *21*
sjFromExons, 15

transmogR *(*transmogR-package*)*, 2
transmogR-package, 2
transmogrify, 17
transmogrify(), *2*, *10*, *13*
transmogrify,BSgenome,GRanges,GRanges-method *(*transmogrify*)*, 17
transmogrify,BSgenome,VcfFile,GRanges-method *(*transmogrify*)*, 17
transmogrify,XStringSet,GRanges,GRanges-method *(*transmogrify*)*, 17
transmogrify,XStringSet,VcfFile,GRanges-method *(*transmogrify*)*, 17

transmogrify-methods *(transmogrify)*, 17

upsetVarByCol, 20
upsetVarByCol(), *3*

VariantAnnotation::ScanVcfParam, *4*, *8*, *19*
VariantAnnotation::VcfFile, *4*, *8*
varTags, 22
varTypes, 23
varTypes(), *3*, *22*