Package 'bacontrees'

May 12, 2026

Title Bayesian Context Trees for Discrete Sequence Data

Version 1.0.0

Description Models discrete sequential data using Bayesian Context Trees. Context trees, also known as Variable Length Markov Chains (VLMCs), are parsimonious Markov models where the order of dependence can vary with the observed past. Provides a generic 'R6' class structure that exposes the full tree for building custom algorithms, exact Bayesian inference via a bottom-up recursive algorithm (closed-form marginal likelihood, Maximum A Posteriori (MAP) tree, exact posterior probabilities, and exact sampling from the posterior), a frequentist estimator via the context algorithm with likelihood-ratio pruning, simulation utilities, and a Metropolis-Hastings sampler. See Paulichen and Freguglia (2026) <doi:10.48550/arXiv.2603.25806>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Imports R6, stringr, glue, purrr, dplyr, progressr, Rcpp, igraph, ggraph, ggplot2, Brobdingnag

Depends R (>= 4.1.0)

LazyData true

LinkingTo Rcpp

NeedsCompilation yes

Author Victor Freguglia [aut, cre] (ORCID: <https://orcid.org/0000-0002-6189-4453>), Thiago Paulichen [ctb] (ORCID: <https://orcid.org/0009-0002-5505-2409>)

Maintainer Victor Freguglia <victorfreguglia@gmail.com>

Repository CRAN

Date/Publication 2026-05-12 19:20:31 UTC

Contents

`abc_data` *Simulated Sequence Data from a Variable Length Markov Chain (VLMC)*

Description

The data objects `abc_vec` and `abc_list` contain sequences generated using a Variable Length Markov Chain (VLMC) with a specified context tree and transition probabilities.

Usage

```
abc_list

abc_vec
```

Format

- `abc_vec`: A character vector of length 1000 representing a single sequence generated by the VLMC.
- `abc_list`: A list of three character vectors, each of length 1000, representing independent sequences generated by the VLMC.

An object of class `character` of length 1000.

Details

The VLMC used to generate the data was based on an alphabet of three symbols (`a`, `b`, and `c`) with a context tree constructed to a maximal depth of 3. The active contexts and their corresponding transition probabilities are as follows:

- `*.a`: `c(0.10, 0.20, 0.70)`
- `*.b`: `c(0.33, 0.33, 0.34)`
- `*.c.a`: `c(0.20, 0.10, 0.70)`
- `*.c.b`: `c(0.01, 0.98, 0.01)`

- *.c.c: c(0.40, 0.40, 0.20)

The sequences were generated using a seed value of 1 to ensure reproducibility.

Source

The data was generated using the rvlmc function in the R package **bacontrees**.

Examples

```
# Access the single sequence
data(abc_vec)
print(abc_vec)

# Access the list of sequences
data(abc_list)
print(abc_list)
```

---

baConTree *Bayesian Context Tree R6 Class*

Description

The baConTree class extends ContextTree to support Bayesian inference, including Dirichlet priors, context prior weights, and Metropolis-Hastings sampling for posterior inference on context trees.

Details

This class provides methods for running MCMC and extracting posterior samples for Bayesian context tree models.

Value

Gets the sampled chain stored.

A numeric (when log = TRUE) or a brob (when log = FALSE).

Invisibly returns the object itself. As a side-effect, the MAP tree is set as the active tree.

A list of length two, containing the prior and posterior probabilities (or their logarithms, if log = TRUE), in this order.

Super class

bacontrees::ContextTree -> baConTree

Methods

Public methods:

- baConTree$new()
- baConTree$runMetropolisHastings()
- baConTree$getChain()
- baConTree$sampleTree()
- baConTree$getMarginalLikelihood()
- baConTree$activateMap()
- baConTree$activeTreeProbabilities()
- baConTree$clone()

Method new():

Usage:

```
baConTree$new(
  data,
  maximalDepth = 5,
  alpha,
  priorWeights,
  initialTree = c("map", "root")
)
```

Arguments:

`data` Either a vector with discrete data or a list of vectors.

`maximalDepth` Depth of the maximal tree considered.

`alpha` Hyperparameter for the Dirichlet prior distribution of probabilities.

`priorWeights` A function to be evaluated at each node that returns its weight in the prior distribution.

`initialTree` Either `"map"` (default) or `"root"`. Controls the initial active tree: `"map"` activates the MAP tree immediately after initialization, while `"root"` starts from the root-only tree.

Method runMetropolisHastings():

Usage:

```
baConTree$runMetropolisHastings(steps)
```

Arguments:

`steps` Number of steps to run the Metropolis Hastings algorithm for.

Details: This method supports progress monitoring via the **progressr** package. Users can wrap the function call in `with_progress()` to display a progress bar while the function executes. If no progress handler is registered, the function will run without showing progress.
To enable progress, register a handler and wrap the function call in `with_progress()`.

Method getChain(): Chain generated via Metropolis Hastings algorithm.

Usage:

```
baConTree$getChain()
```

Method `sampleTree()`**:** Samples a context tree exactly from the prior or posterior distribution using the per-node branching probabilities.

Usage:

```
baConTree$sampleTree(type = c("prior", "posterior"))
```

Arguments:

`type` Either `"prior"` or `"posterior"`. Determines which branching probabilities are used for sampling. `"prior"` uses `priorBranchingProbability` and `"posterior"` uses `posteriorBranchingProbability`.

Details: The algorithm starts from the root-only tree and at each non-leaf node the tree is grown (via `growActive`) with probability equal to the node's branching probability; otherwise the node remains a leaf of the sampled tree and its subtree is never visited. This yields an exact sample from the prior or posterior distribution over context trees.
The branching probability at a node is `prod(children sigmaPrior) / sigmaPrior` for the prior, and the analogous quantity for the posterior, so that it equals one minus the probability that the process stops at that node.

Returns: Returns the active tree code of the sampled tree (a character string as produced by `activeTreeCode()`). As a side-effect, the object's active tree is set to the sampled tree.

Method `getMarginalLikelihood()`**:** Marginal likelihood of the data under the Bayesian context tree model.

Usage:

```
baConTree$getMarginalLikelihood(log = TRUE)
```

Arguments:

`log` Logical. If `TRUE` (default), returns the log marginal likelihood as a plain R `numeric`. If `FALSE`, returns the marginal likelihood as a `brob` object.

Details: This is computed from `sigmaPosterior/sigmaPrior` at the root node, which equals the sum over all trees of `prior(T) * p(data | T)`, normalised by the prior partition function.

Method `activateMap()`**:** Activates the smallest Maximum a Posteriori (MAP) tree under the Bayesian context tree model.

Usage:

```
baConTree$activateMap()
```

Details: Starting from the root node (initially active) and proceeding recursively through its descendants. The method compares the posterior weight at the node with the maximum posterior value attainable over all sub-tree configurations below it. If stopping at the node achieves the maximum posterior (`node$extra$isMapLeaf = TRUE`) the node remains active and the recursion stops along that branch (i.e., the sub-tree below the node is pruned). Otherwise, the node is deactivated, its children are activated, and the procedure continues recursively.

Method `activeTreeProbabilities()`**:** Computes the prior and posterior probabilities of the active tree under the Bayesian context tree model.

Usage:

```
baConTree$activeTreeProbabilities(log = FALSE)
```

Arguments:

log Logical. If FALSE (default), returns the prior and posterior probabilities of the active tree. If TRUE, returns their logarithms.

Details: The probabilities are obtained by taking the product of the weights associated with all active nodes in the tree. These quantities are then normalized by the corresponding normalizing constants, namely the value of sigmaPrior at the root node for the prior and the value of sigmaPosterior at the root for the posterior.
For numerical stability, all computations are performed on the logarithmic scale, which avoids underflow when dealing with small values.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
baConTree$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
bt <- baConTree$new(abc_list, maximalDepth = 3, alpha = 0.01,
                    priorWeights = function(node) exp(-1/3*node$getDepth()))
bt$runMetropolisHastings(300)
chain <- bt$getChain()
```

ContextTree *ContextTree R6 Class*

Description

The ContextTree class represents a variable-length Markov context tree, supporting construction, manipulation, and data assignment for context tree models. It manages nodes, active/inactive states, and provides methods for growing, pruning, and validating the tree.

Details

This class is the core data structure for context tree modeling, supporting both root and maximal initialization, and efficient management of tree structure and data.

Active bindings

nodes List of nodes from a context tree (both active and non-active). Read-only.

Methods

Public methods:

- ContextTree$new()
- ContextTree$root()
- ContextTree$validate()
- ContextTree$getDataset()
- ContextTree$getAlphabet()
- ContextTree$getMaximalDepth()
- ContextTree$getActiveNodes()
- ContextTree$activeTreeCode()
- ContextTree$activateRoot()
- ContextTree$activateMaximal()
- ContextTree$activateByCode()
- ContextTree$activateFromContexts()
- ContextTree$getLeaves()
- ContextTree$getInnerNodes()
- ContextTree$nodeExists()
- ContextTree$getParentNode()
- ContextTree$getChildrenNodes()
- ContextTree$getSiblingNodes()
- ContextTree$growActive()
- ContextTree$pruneActive()
- ContextTree$getGrowableNodes()
- ContextTree$getPrunableNodes()
- ContextTree$setData()
- ContextTree$igraph()
- ContextTree$print()
- ContextTree$clone()

Method `new()`**:** Initializes a `ContextTree` object with a given maximal depth. If `dataset` is provided, the alphabet is inferred from data.

Usage:

```
ContextTree$new(dataset = NULL, maximalDepth = 3, alphabet = NULL)
```

Arguments:

`dataset` A `Sequence` object, a character vector with a single observed chain or a list of vectors of observed chains to be set as data for the context tree.

`maximalDepth` Depth of the maximal tree considered.

`alphabet` Either an object of class Alphabet or a character vector with the symbols for the alphabet considered in the context tree.

Method `root()`**:**

Usage:

```
ContextTree$root()
```

Returns: Returns the Context Tree root node.

Method `validate()`**:**

Usage:

```
ContextTree$validate()
```

Returns: Returns a logical value incating whether the Context Tree is valid.

Method `getDataset()`**:**

Usage:

```
ContextTree$getDataset()
```

Returns: Returns the dataset as a `Sequence` object.

Method `getAlphabet()`**:**

Usage:

```
ContextTree$getAlphabet()
```

Returns: Returns the alphabet related to the Context Tree.

Method `getMaximalDepth()`**:**

Usage:

```
ContextTree$getMaximalDepth()
```

Returns: Returns the maximal depth of the tree.

Method `getActiveNodes()`**:**

Usage:

```
ContextTree$getActiveNodes(idx = TRUE)
```

Arguments:

`idx` A logical value. If TRUE, the function will return the index (path) of the node as a string. If FALSE, returns a list of nodes.

Returns: Returns a list of active nodes (leaf nodes of the active tree).

Method `activeTreeCode()`**:**

Usage:

```
ContextTree$activeTreeCode()
```

Returns: Returns a character value representing the active tree.

Method `activateRoot()`**:** Sets the active tree to be the one containing only the root node.

Usage:

```
ContextTree$activateRoot()
```

Method `activateMaximal()`**:** Activates the leaf nodes of the maximal Context Tree.

Usage:

```
ContextTree$activateMaximal()
```

Method `activateByCode()`**:** Sets the active tree to be the one corresponding to a tree code obtained from the `activeTreeCode` method.

Usage:

```
ContextTree$activateByCode(code)
```

Arguments:

`code` The tree code for the tree to be activated.

`code` The tree code for the tree to be activated.

Method `activateFromContexts()`**:** Sets the active tree to match a specified set of contexts, given as a character vector or a brace-enclosed comma-separated string. The contexts must be compatible with the tree's existing alphabet and maximal depth.

Usage:

```
ContextTree$activateFromContexts(contexts)
```

Arguments:

`contexts` Character vector or string. A vector of context paths (e.g., `c("*.a", "*.b.a", "*.b.b")`) or a single brace-enclosed string (e.g., `"\{*.a, *.b.a, *.b.b\}"`).

Method `getLeaves()`**:**

Usage:

```
ContextTree$getLeaves(idx = TRUE)
```

Arguments:

`idx` A logical value. If TRUE, the function will return the index (path) of the node as a string. If FALSE, returns a list of nodes.

Returns: Returns the leaf nodes of the maximal Context Tree (regardless of the current active tree).

Method `getInnerNodes()`**:**

Usage:

```
ContextTree$getInnerNodes(idx = TRUE)
```

Arguments:

`idx` A logical value. If TRUE, the function will return the index (path) of the node as a string. If FALSE, returns a list of nodes.

Returns: Returns the inner nodes nodes of the active Context Tree. Inner nodes are nodes that are in the path between the root (including) and active nodes.

Method `nodeExists()`**:**

Usage:

```
ContextTree$nodeExists(path)
```

Arguments:

`path` A string representing the path of a node.

Returns: TRUE if a node with a given path exists in the Context Tree.

Method `getParentNode()`**:**

Usage:

```
ContextTree$getParentNode(path, idx = TRUE)
```

Arguments:

`path` A string representing the path of a node.

`idx` A logical value. If TRUE, the function will return the index (path) of the node as a string. If FALSE, returns a list of nodes.

Returns: Returns the parent node of the node in a given path.

Method `getChildrenNodes()`**:**

Usage:

```
ContextTree$getChildrenNodes(path, idx = TRUE)
```

Arguments:

`path` A string representing the path of a node.

`idx` A logical value. If TRUE, the function will return the index (path) of the node as a string. If FALSE, returns a list of nodes.

Returns: Returns the parent node of the node in a given path.

Method `getSiblingNodes()`**:**

Usage:

```
ContextTree$getSiblingNodes(path, idx = TRUE)
```

Arguments:

`path` A string representing the path of a node.

`idx` A logical value. If TRUE, the function will return the index (path) of the node as a string. If FALSE, returns a list of nodes.

Returns: Returns the sibling nodes of the node in a given path.

Method `growActive()`**:** Replaces the node of a given path by its children in the active tree.

Usage:

```
ContextTree$growActive(path)
```

Arguments:

`path` A string representing the path of a node.

Method `pruneActive()`**:** Replaces the node of a given path and its siblings by the parent node in the active tree.

Usage:

```
ContextTree$pruneActive(path)
```

Arguments:

`path` A string representing the path of a node.

Method `getGrowableNodes()`**:**

Usage:

```
ContextTree$getGrowableNodes()
```

Returns: Returns a list of paths of active nodes that have children that can be activated.

Method `getPrunableNodes()`**:**

Usage:

```
ContextTree$getPrunableNodes()
```

Returns: Returns a list of paths of active nodes that have all siblings active.

Method `setData()`**:** Sets data for the Context Tree by setting the counts of occurrences of each symbol of the alphabet within each context (node) of the tree.

Usage:

```
ContextTree$setData(dataset)
```

Arguments:

`dataset` A `Sequence` object, a character vector with a single observed chain or a list of vectors of observed chains to be set as data for the context tree.

Method `igraph()`**:**

Usage:

```
ContextTree$igraph(activeOnly = TRUE)
```

Arguments:

`activeOnly` logical value. If TRUE, only the nodes in the active tree are plotted (including internal nodes).

Details: Converts the ContextTree to an `igraph` object. All attributes in the `extra` field of nodes are included in the attributes of each node for the `igraph`.

Returns: Returns an igraph object.

Method `print()`**:** Prints the current active Context Tree and the counts for each context.

Usage:

```
ContextTree$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ContextTree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
tree <- ContextTree$new(abc_list, maximalDepth = 3)
tree$activateMaximal()
print(tree)
```

`fit_vlmc` *Fit a Variable Length Markov Chain (VLMC) via Context Algorithm*

Description

Fits a VLMC model to discrete sequence data using the context algorithm, performing likelihood ratio tests to prune the context tree.

Usage

```
fit_vlmc(data, cutoff = 10, max_length = 6)
```

Arguments

`data`	Either a character vector (single sequence) or a list of character vectors (multiple sequences).
`cutoff`	Numeric. Cutoff value for the (log) likelihood ratio test statistic used for pruning.
`max_length`	Integer. Depth of the maximal tree considered.

Details

The function builds a maximal context tree, computes log-likelihoods for each node, and prunes nodes whose likelihood ratio test statistic is below the cutoff. The result is a pruned context tree representing the fitted VLMC.

Value

A `ContextTree` object representing the fitted VLMC as the active tree.

Examples

```
data(abc_list)
fit <- fit_vlmc(abc_list, cutoff = 20, max_length = 4)
print(fit$getActiveNodes())
```

`metropolis_vlmc` *Metropolis-Hastings Posterior Sampling for Context Trees*

Description

Runs the Metropolis-Hastings algorithm to sample from the posterior distribution of context trees given sequence data, Dirichlet priors, and context prior weights.

Usage

```
metropolis_vlmc(
  data,
  n_steps,
  max_depth = 6,
  alpha = 0.001,
  context_weights = function(node) 1,
  burnin = 100,
  thin = 1
)
```

Arguments

`data`	Either a character vector (single sequence) or a list of character vectors (multiple sequences).
`n_steps`	Integer. Number of MCMC steps to run.
`max_depth`	Integer. Maximum depth for the context tree.
`alpha`	Numeric. Dirichlet prior parameter for transition probabilities.
`context_weights`	Function. Returns the prior weight for a given node.
`burnin`	Integer. Number of initial iterations to discard from posterior summaries.
`thin`	Integer. Thinning interval for posterior summaries.

Details

The Markov chain is initialized from the maximal tree (all leaves active).

This function supports progress monitoring via the **progressr** package. Wrap the call in `with_progress()` to display a progress bar.

Value

An object of class `metropolis_vlmc` with elements:

- `df`: Data frame of context sets, tree codes, counts, and posterior probabilities.
- `codes`: List of context sets for each tree code.
- `chain`: The full chain of tree codes sampled.

Examples

```
data(abc_list)
result <- metropolis_vlmc(abc_list, n_steps = 1000, max_depth = 3)
print(result)
```

`plot.baConTree` *Plot method for baConTree class*

Description

Plot method for baConTree class

Usage

```
## S3 method for class 'baConTree'
plot(x, ...)
```

Arguments

`x`	A `baConTree` object.
`...`	Not used.

Details

Plots the full maximal tree. Each node is labelled and filled according to its posterior branching probability, i.e. the probability that the node has children in the true context tree given the data. Values close to 1 (dark blue) indicate near-certain branching; values close to 0 (white) indicate the node is likely a leaf. Active edges are drawn solid and inactive edges dotted, matching the convention of `plot.ContextTree`.

The plot is done using ggraph and can be further modified.

`plot.ContextTree` *Plot method for ContextTree class*

Description

Plot method for ContextTree class

Usage

```
## S3 method for class 'ContextTree'
plot(x, ..., activeOnly = TRUE)
```

Arguments

x	A ContextTree object.
...	Not used.
activeOnly	logical value. If TRUE, only the nodes in the active tree are plotted (including internal nodes).

Details

The edges of the tree have their width corresponding to the number of occurrences of contexts matching the child node.

The plot is done using ggraph and can be further modified.

---

rvlmc *Generate a sequence based on a Variable Length Markov Chain (VLMC)*

Description

This function simulates one or more sequences using a Variable Length Markov Chain (VLMC) model, where the memory (context) length can vary depending on the context tree provided.

Usage

```
rvlmc(n, alphabet, context_list, context_probs)
```

Arguments

n	An integer specifying the length of the sequence to generate, or a vector of lengths for multiple sequences.
alphabet	A character vector representing the set of symbols used in the sequence.
context_list	A character vector specifying the contexts of the VLMC. Each context is a dot-separated string starting with "*" (the root sentinel), where subsequent symbols represent the observed past in **reverse chronological order** (most recent first). For example, "*.c.b" denotes the context where the last observed symbol was "c" and the one before it was "b". The set of contexts must form a **complete partition** of all possible pasts (i.e., a valid full context tree); the function will error if this is not satisfied.
context_probs	A list of numeric probability vectors, one per element of context_list, in the same order. Each vector must have the same length as alphabet, with elements giving the probability of sampling the corresponding symbol in alphabet.

Details

The function generates sequences by traversing the context tree defined by `context_list` and `context_probs`. For each position in the sequence, the most specific matching context in `context_list` is used to sample the next symbol according to the corresponding probability vector.

The first `H` symbols (where `H` is the depth of the deepest context) are sampled uniformly from `alphabet` as an initialisation step and are not drawn from the VLMC model.

Value

If `n` is a single integer, returns a character vector of length `n` representing the generated sequence. If `n` is a vector of length > 1, returns a list of character vectors, each of the specified length.

Examples

```
# Define parameters for the VLMC
n <- 1000
alphabet <- c("a", "b", "c")
context_list <- c("*.a", "*.b", "*.c.a", "*.c.b", "*.c.c")
context_probs <- list(
  c(0.10, 0.20, 0.70), # for *.a
  c(0.33, 0.33, 0.34), # for *.b
  c(0.20, 0.10, 0.70), # for *.c.a
  c(0.01, 0.98, 0.01), # for *.c.b
  c(0.40, 0.40, 0.20)  # for *.c.c
)

# Generate a single sequence
sequence <- rvlmc(n, alphabet, context_list, context_probs)
print(sequence)

# Generate multiple sequences of different lengths
n_vec <- c(100, 200, 150)
sequences <- rvlmc(n_vec, alphabet, context_list, context_probs)
str(sequences)
```

Sequence *Sequence Class*

Description

The Sequence class provides a unified representation of one or more sequences of categorical symbols. When initialized, the input is encoded as integers according to a user-defined or automatically inferred alphabet.

Details

The class accepts:

- a **list** of character or numeric vectors (multiple sequences)
- a **single** character or numeric vector (one sequence)

The alphabet can be provided manually or inferred via `infer_alphabet()`, which must return an object containing a field `symbols` defining the valid symbol set.

Internally, all sequences are stored as integer vectors using `as.numeric(factor(..., levels = Alphabet$symbols))`.

Value

An R6 object of class `Sequence`.

Fields

`data` A list of integer-encoded sequences.

`n` A vector with the length of each sequence.

`Alphabet` An object defining the symbol set used for encoding.

Methods

`initialize(x, alphabet = NULL)` Create a new object from input.

`print()` Display a compact summary.

`TreeNode` *TreeNode Class*

Description

This class represents a tree node with a path, depth, and various methods for managing its state and validating its path.

Public fields

`extra` List to hold extra information.

Active bindings

`counts` Numeric vector to hold counts (active binding; validated on write).

Methods

Public methods:

- TreeNode$new()
- TreeNode$print()
- TreeNode$activate()
- TreeNode$deactivate()
- TreeNode$isActive()
- TreeNode$getDepth()
- TreeNode$isLeaf()
- TreeNode$getPath()
- TreeNode$getParentPath()
- TreeNode$getChildrenPaths()
- TreeNode$validatePath()
- TreeNode$clone()

Method `new()`**:** Initializaes the `TreeNode`

Usage:

```
TreeNode$new(path)
```

Arguments:

`path` A character string representing the path of the node.

Method `print()`**:** Print the path of the node

Usage:

```
TreeNode$print(...)
```

Arguments:

`...` Additional arguments passed to the `cat` function.

Method `activate()`**:** Activate the node.
This method sets the node's active status to TRUE.

Usage:

```
TreeNode$activate()
```

Method `deactivate()`**:** Deactivate the node.
This method sets the node's active status to FALSE.

Usage:

```
TreeNode$deactivate()
```

Method `isActive()`**:**

Usage:

```
TreeNode$isActive()
```

Returns: Logical value indicating if the node is active.

Method `getDepth()`**:**

Usage:

```
TreeNode$getDepth()
```

Returns: Integer value representing the depth of the node.

Method `isLeaf()`**:**

Usage:

```
TreeNode$isLeaf()
```

Returns: Returns a logical value indicating whether the node is a leaf (of the maximal tree, not the active tree).

Method `getPath()`**:**

Usage:

```
TreeNode$getPath()
```

Returns: Character string representing the path of the node.

Method `getParentPath()`**:**

Usage:

```
TreeNode$getParentPath()
```

Returns: Character string representing the parent path of the node.

Method `getChildrenPaths()`**:**

Usage:

```
TreeNode$getChildrenPaths()
```

Returns: Character string representing the paths for the children of a node.

Method `validatePath()`**:** Validate the node path against an alphabet
This method validates the node's path by checking if all elements of the path (excluding the first element) are in the provided alphabet.

Usage:

```
TreeNode$validatePath(Alphabet)
```

Arguments:

`Alphabet` An `Alphabet` object containing a `symbols` vector to validate against.

Returns: Logical value indicating if the path is valid.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TreeNode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index