# Package ‘autoharp’

February 19, 2026

**Title** Semi-Automatic Grading of R and Rmd Scripts

**Version** 0.2.0

**Description** A customisable set of tools for assessing and grading R or R-markdown scripts from students. It allows for checking correctness of code output, runtime statistics and static code analysis. The latter feature is made possible by representing R expressions using a tree structure.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Imports** magrittr, dplyr, stringr, rlang, tibble, knitr, rmarkdown, shiny, lintr, methods, igraph, xfun

**Collate** 'treeharp.R' 'th_getter-length.R' 'as.matrix.R' 'autoharp.R' 'check_correctness.R' 'check_rmd.R' 'check_runtime.R' 'count_lints.R' 'env_size.R' 'extracton_section_text.R' 'forestharp.R' 'forestharp_helpers.R' 'generate_thumbnails.R' 'join_treeharp.R' 'lang_2_tree.R' 'lang_2_tree_helpers.R' 'log_summary.R' 'nlp_related.R' 'populate_soln_env.R' 'pre_checks.R' 'render_one.R' 'reset_path.R' 'to_BFS.R' 'tree_kernel.R' 'tree_routines.R' 'utils-pipe.R' 'utils.R' 'write_html.R' 'zzz.R'

**Suggests** testthat

**SystemRequirements** pandoc (>= 1.14) - http://pandoc.org

**URL** https://singator.github.io/autoharp-docs/

**NeedsCompilation** no

**Author** Vik Gopal [aut],
Agrawal Naman [aut, cre],
Samuel Seah [aut],
Viknesh Jeya Kumar [aut],
Gabriel Ang [aut],
Ruofan Liu [ctb],
National University of Singapore [cph]

**Maintainer** Agrawal Naman <naman.a@nus.edu.sg>

**Depends** R (>= 3.5.0)

**Repository** CRAN

**Date/Publication** 2026-02-19 11:30:02 UTC

# Contents

adj_list_2_matrix . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
apply_can_improve_this . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
as.matrix . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
autoharp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
carve_mst . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
carve_subtree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
check_correctness . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
check_rmd . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
check_runtime . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
clean_dir . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
copy_e2e . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
count_lints_all . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
count_lints_one . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
env_size . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
extract_chunks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
extract_non_chunks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
extract_section_text . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
fapply . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
find_branch_num . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
forestharp-helpers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
generate_all_subtrees . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
generate_thumbnails . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
get_adj_list . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
get_child_ids . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
get_child_ids2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
get_levels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
get_libraries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
get_next_depth_id . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
get_next_subtree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
get_node_types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
get_parent_call_id . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
get_parent_id . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
get_parent_id2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
get_recursive_index . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
is_connected . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
is_cyclic_r . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
is_subtree_rooted_at . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
jaccard_treeharp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
join_treeharps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
K2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

keep_branches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
lang_2_tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
log_summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
matrix_2_adj_list . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
only_actual_args_differ . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
path_to_root . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
pipe_can_improve_this . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
plot,TreeHarp-method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
populate_soln_env . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
prune_depth . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
rbind_to_nodes_info . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
render_one . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
render_prechecks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
replace_sp_chars_filename . . . . . . . . . . . . . . . . . . . . . . . . . . 44
reset_path . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
rmd_to_forestharp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
rmd_to_token_count . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
subtree_at . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
to_BFS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
TreeHarp-class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
tree_sim . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
update_adj_list . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52

**Index** **53**

---

## `adj_list_2_matrix` *Convert adjacency list to a matrix*

---

### Description

Converts a list that represents a tree into a binary matrix.

### Usage

```
adj_list_2_matrix(adj_list)
```

### Arguments

| | |
|---|---|
| `adj_list` | The adjacency list of the tree. |

### Details

Remember that the list has to be for a tree, not a general graph. Please see other help pages for more specifications.

This is a low-level function, used within the S4 class TreeHarp. It is not generally meant for use by the user.

It works by filling up the upper diagonal of the matrix before reflecting it.

### Value

A symmetric matrix of 1's and 0's, with 1 in entry (i,j) representing an edge between the two vertices.

---

## apply_can_improve_this

*Identifies if an apply function can improve the code*

---

### Description

Identifies if an apply function can improve the code

### Usage

```
apply_can_improve_this(fname, window_size = 2)
```

### Arguments

| | |
|---|---|
| `fname` | A filename: either Rmd/qmd or R. |
| `window_size` | A window size to analyse. |

### Details

First, all library() calls and read() calls are removed. Next, `only_actual_args_differ` is run on all pairwise expressions. The matrix is searched for blocks of 1's along the diagonal.

### Value

A numeric vector corresponding to windows for review is returned. If none are found, then NULL is returned.

---

## as.matrix

*TreeHarp Cast a TreeHarp to Matrix.*

---

### Description

Convert a treeharp object to an adjacency matrix.

### Arguments

| | |
|---|---|
| `from` | A treeharp object. |

### Value

A matrix.

`autoharp` *autoharp: Semi-Automatic Grading of R and Rmd Scripts*

## Description

The autoharp package provides functions for running and analysing R script/Rmd submissions from students.

## Main functions are

1. `populate_soln_env`
2. `render_one`
3. `TreeHarp`

The user manuals can be found at `https://singator.github.io/autoharp-docs/`

## Author(s)

**Maintainer**: Agrawal Naman <`naman.a@nus.edu.sg`>

Authors:

- Vik Gopal <`vik.gopal@nus.edu.sg`>
- Samuel Seah <`samuelseah@u.nus.edu`>
- Viknesh Jeya Kumar <`viknesh@u.nus.edu`>
- Gabriel Ang <`gabrielang@u.nus.edu`>

Other contributors:

- Ruofan Liu <`kelseyliu1998@gmail.com`> [contributor]
- National University of Singapore [copyright holder]

## See Also

Useful links:

- `https://singator.github.io/autoharp-docs/`

`carve_mst` *Carve a Minimal Spanning Tree Out*

## Description

Given node names, this function retrieves the smallest tree containing at most those nodes.

## Usage

```
carve_mst(th, node_names)
```

## Arguments

| | |
|---|---|
| `th` | An object of class TreeHarp. |
| `node_names` | A character vector of node names. Nodes outside this set will not be returned in the tree. It must include the root node name. |

## Details

The function starts from each node specified and works it's way up to the root. If a branch contains nodes outside the list, it is shortened.

In the end, the tree that is returned will try to contain all the named nodes, but if that's not possible some will dropped to ensure a tree is returned, not a disconnected graph.

## Value

An object of class TreeHarp.

## Examples

```
ex1 <- quote(x <- f(y, g(5)))
th1 <- TreeHarp(ex1, TRUE)
carve_mst(th1, c("<-", "x", "f", "5")) ## note: 5 is dropped.
carve_mst(th1, c("<-", "x", "f", "y"))
carve_mst(th1, c("<-", "f", "g"))
```

## carve_subtree *Carve out branches to form a new tree.*

### Description

This functions keeps only the indicated nodes, returning a new sub-tree.

### Usage

```
carve_subtree(obj, char_arr)
```

### Arguments

| | |
|---|---|
| `obj` | An object of class TreeHarp. |
| `char_arr` | A vector of 1's and 0's indicating which nodes to keep. The vector should have length equal to the number of nodes in obj. |

### Details

This returns an error if the sub-tree does not define a new tree.

### Value

An object of class TreeHarp.

### Examples

```
th3 <- list(a= c(2L,3L,4L), b=NULL, c=c(5L, 6L), d=7L, e=NULL, f=NULL, g=NULL)
carve_subtree(TreeHarp(th3), c(1,0,0,0,0,0,0))
st <- subtree_at(TreeHarp(th3), 4)
plot(st)
```

## check_correctness *Check correctness of student solution rmd.*

### Description

This will run unit tests on the students' rmd file.

### Usage

```
check_correctness(e_stud, e_soln, test_fname)
```

### Arguments

| | |
|---|---|
| `e_stud` | The environment containing the output objects from running the studnent Rmd file. |
| `e_soln` | The environment containing the objects from the solution template. It will probably contain objects with the suffix "_soln". These will be tested against the versions generated by the student. |
| `test_fname` | The R script containing the test chunks. |

### Details

Prior to calling this, `populate_soln_env` should already have been called on the solution template, and the student file should already have been knitted in order to generate the students' objects. Of course, one could generate the test script independent of `populate_soln_env`, but the solution environment that contains objects with a "_soln" suffix is also needed.

The student environment, solution environment, test file and the list of tests and expectations are the inputs to this function.

### Value

A data frame with one row, and the number of columns equal to the number of tests run plus the number of scalars to keep.

### See Also

`populate_soln_env`, `render_one`

---

## `check_rmd` *Check if a File is Rmd*

---

### Description

Checks if a file actually is an Rmd file.

### Usage

```
check_rmd(fname, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `fname` | A character string. It is the name of the student submission file. |
| `verbose` | A logical value that prints messages if a non-rmd file is found. |

### Details

It runs three checks. First, it checks for the file extension to be Rmd or rmd or any such variant. Second, it checks for a YAML header at the beginning of file. Finally, it checks if there is at least one properly defined R chunk within the file.

### Value

The function will return TRUE if all the (3) checks pass, and FALSE otherwise.

### See Also

`get_libraries`

---

## `check_runtime` *Calculate Run-time Statistics*

---

### Description

This is stand-alone function. It computes the runtime stats without rendering the md/html/pdf file.

### Usage

```
check_runtime(stud_fname, knit_root_dir, return_env = FALSE)
```

### Arguments

| | |
|---|---|
| `stud_fname` | The rmd filename of the student. |
| `knit_root_dir` | The working directory to use when knitting the file. |
| `return_env` | A logical value to indicate if the environment from the rmd file should be return. If FALSE, an NA value is returned. |

### Details

This routine is not used within any other function within the package. Figures are not cleaned or removed.

### Value

A list containing the running time in seconds, the memory used by the final environment in bytes (as a numeric scalar), and the environment object containing all the generated objects from the rmd file.

### See Also

`render_one`

## `clean_dir` *Removes md Files when no HTML Present*

### Description

Cleans up the autoharp output directory.

### Usage

```
clean_dir(dir_name, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `dir_name` | The directory containing the files to be cleaned. |
| `verbose` | If TRUE, then the files and directories being removed will be printed. |

### Details

When batch rendering Rmd files, it is inevitable that some files fail. These files would have their knit.md and utf.md present, but they would not have a corresponding html file generated.

This function is called for its' side-effect, to remove those lonely md files.

If this clean-up is not done, when we try to re-run the files (perhaps with some of the errors fixed), these straggling md files will cause problems. The most crucial one is that the Rmd files will not be re-knitted, even though they have been changed.

### Value

No return value.

## `copy_e2e` *Copy an object from one env to another.*

### Description

A wrapper function that uses assign and get.

### Usage

```
copy_e2e(from_obj, from_env, to_obj, to_env)
```

### Arguments

| | |
|---|---|
| `from_obj` | The name of the object to copy. It has to be a string. |
| `from_env` | The environment in which the object lives. It has to be an object of class environment. |
| `to_obj` | The name of the object to assign it to, in the new environment. Also a string. |
| `to_env` | The environment to which the new object is to be assigned. It has to be an object of class environment. |

### Value

There is no return value. This function is called for its' side effect.

### Examples

```
e1 <- new.env(); e2 <- new.env()
ls(e2)
evalq(x <- 1L, e1)
copy_e2e("x", e1, "y", e2)
ls(e2)
```

---

## `count_lints_all` *Lint counter*

---

### Description

Count number of lints in one folder

### Usage

```
count_lints_all(file_names, lint_list, lint_labels)
```

### Arguments

| | |
|---|---|
| `file_names` | The path to the rmd files that need to be checked for lints. |
| `lint_list` | List of lints to check for. |
| `lint_labels` | List of labels to name the vector to return. |

### Details

The function will count the number of lints in a file. The lints to be checked can be passed as an argument. Else, the default will be used. The defaults are as follows:

- T_and_F_symbol_linter
- line_length_linter
- assignment_linter
- absolute_path_linter
- pipe_continuation_linter

Note that labels would also need to be given if the non-default lints are chosen.

### Value

Dataframe containing the lints.

---

## `count_lints_one` *File lint counter*

---

### Description

Count number of lints in one file

### Usage

```
count_lints_one(rmd_file, lint_list, lint_labels)
```

### Arguments

| | |
|---|---|
| `rmd_file` | The path to the rmd file to check for lints. |
| `lint_list` | List of lints to check for. |
| `lint_labels` | List of labels to name the vector to return. |

### Details

The function will count the number of lints in a file. The lints to be checked can be passed as an argument. Else, the default will be used. The defaults are as follows: * T_and_F_symbol_linter * line_length_linter * assignment_linter * absolute_path_linter * pipe_continuation_linter Note that labels would also need to be given if the non-default lints are chosen.

### Value

Vector containing the lints.

---

## `env_size` *Calculates the Total Memory Used*

---

### Description

This function uses the utils package to compute the total amount of memory used by objects in an environment.

### Usage

```
env_size(env)
```

### Arguments

| | |
|---|---|
| `env` | The environment whose size is to be computed. |

### Details

The names are wrapped in backticks. Otherwise, non-syntactic names will cause problems. This function is used within `render_one` as part of the runtime stats assessment.

### Value

The size in bytes, as a numeric value (scalar).

### Examples

```
e1 <- new.env()
env_size(e1)
evalq(x <- 1:10000L, e1)
env_size(e1)
```

---

## `extract_chunks` *Extract chunks that match a pattern.*

---

### Description

Extracts chunks whose labels match a pattern from the rmd file.

### Usage

```
extract_chunks(rmd_name, pattern)
```

### Arguments

| | |
|---|---|
| `rmd_name` | A character string, the name of the rmd file to get the chunks from. |
| `pattern` | The pattern to match within the label. In fact, the match is applied to the whole chunk option. |

### Value

A list of character vectors. Each vector contains the chunk from the file. If no pattern is specified, all chunks are returned. Remember that the chunk header and tail are also included in the returned list.

## extract_non_chunks *Extract non-chunks from an Rmd file.*

### Description

Extracts non-chunks from an Rmd file.

### Usage

```
extract_non_chunks(rmd_name, out_name)
```

### Arguments

| | |
|---|---|
| rmd_name | A character string, the name of the rmd file to get the chunks from. |
| out_name | An output filename, to dump the text to. |

### Value

If out_name is missing, then a character vector is returned. If outfname is specified, then nothing is returned. The text is written to the file instead.

## extract_section_text *Extract section text from Rmd*

### Description

Extract section text from Rmd

### Usage

```
extract_section_text(rmd_name, hdr_pat, ignore_case = TRUE)
```

### Arguments

| | |
|---|---|
| rmd_name | The filename of the Rmd script. |
| hdr_pat | The regular expression pattern to pick up the section title. |
| ignore_case | A boolean - whether or not to ignore case when matching for the section title. |

### Details

The text that is picked up begins with the specified section, and ends with the next string of pound symbols (#)

## Value

Returns a character vector containing all the text written in the section that begins with the specified pattern.

The pattern should pick up a unique section/sub-section/sub-sub-section. Otherwise, it will stop and raise an error.

---

`fapply` *Apply a function to a forest of trees.*

---

## Description

A convenience function, for applying a function to many trees.

## Usage

```
fapply(fharp, TFUN, combine = TRUE, combiner_fn, ...)
```

## Arguments

| | |
|---|---|
| `fharp` | The output of rmd_to_forestharp. It could also just be a list of TreeHarp objects. |
| `TFUN` | A function that works on a single TreeHarp and returns an output. See forestharp-helpers for examples. |
| `combine` | A logical value that indicates if the output from all function applications should be combined. |
| `combiner_fn` | A function to use to combine the individual output from each tree into a single scalar for each forest. It should handle NA values in the input vector or list. If it is missing, it defaults to sum, with na.rm=TRUE. |
| `...` | Additional arguments to be passed on to TFUN. |

## Details

The input is simply a list of TreeHarp objects. First, the TFUN function is lapply-ed to each Tree-Harp item, resulting in either a list, or a vector with possible NA elements.

The combiner function should be aware of this sort of output, and summarise the list or vector accordingly, handling NA's and returning a *scalar*.

If you need to create a partial function out of a forestharp helper, use an anonymous function, as shown in the examples below.

## Value

A vector, list or a single value. If TFUN returned an error for a particular TreeHarp, that component in the list or vector would be NA. This input vector or list will then be combined by combiner_fn.

## Examples

```
ex1 <- quote(X <- rnorm(10, mean=0.9, sd=4))
ex2 <- quote(Y <- rbeta(10, shape1=3, shape2=5))
f1 <- lapply(c(ex1, ex2), TreeHarp, quote_arg=TRUE)

# returns all function calls that begin with "r", like rnorm and rbeta.
# calls are returned as a list.
fapply(f1, extract_fn_call, combine =FALSE, pattern="^r.*")

# list is catenated.
fapply(f1, extract_fn_call, combine =TRUE, pattern="^r.*",
       combiner_fn = function(x) {paste0(unlist(x), collapse=",")})
```

---

`find_branch_num` *Find the branch that leads from one node to another.*

---

## Description

Given two nodes that are on the same path to the root, this function determines the branch that leads to the child node.

## Usage

```
find_branch_num(th, child_id, ancestor_id)
```

## Arguments

| | |
|---|---|
| `th` | A TreeHarp object. |
| `child_id` | An integer node id. It corresponds to the node to trace up from. |
| `ancestor_id` | An integer node id. It corresponds to the node to trace down from. |

## Details

This is used when trying to find a sub-call from a TreeHarp object. It is useful in determining the indices to use when extracting the sub-call.

## Value

An integer that denotes the branch to follow down (from the ancestor) to reach the child.

## Examples

```
ex3 <- quote(x <- f(y = g(3, 4), z=1L))
t1 <- TreeHarp(ex3, TRUE)
find_branch_num(t1, 8, 3) # should be 1
find_branch_num(t1, 5, 3) # should be 2
```

`forestharp-helpers` *Forestharp helpers*

## Description

Example of functions that can be *directly used on TreeHarp objects individually,* and on forestharp objects via `fapply`.

## Usage

```
count_self_fn(th)

count_lam_fn(th)

count_fn_call(th, pattern, pkg_name)

extract_fn_call(th, pattern, pkg_name)

extract_formal_args(th, fn_name)

extract_assigned_objects(th)

extract_actual_args(th, include_assigned_obj = TRUE)

count_num_lines_for_loop(th)

detect_growing(th, count = FALSE, within_for = FALSE)

detect_for_in_fn_def(th, fn_name)

count_fn_in_fn(th, fn_name, sub_fn)

detect_fn_call_in_for(th, fn_name)

extract_self_fn(th)

detect_fn_arg(th, fn_name, arg)

detect_nested_for(th)
```

## Arguments

| | |
|---|---|
| `th` | A TreeHarp object. |
| `pattern` | A regular expression to pick up function names. |
| `pkg_name` | The name of a package to match functions with. This should be an exact match for the package name. The package should be attached for this to work. In order |

| | |
|---|---|
| | to avoid picking up duplicate names, for instance `tolower` is a function in base R and in ggplot2, run `get_libraries` on the file as well, and match against it. |
| `fn_name` | Function name, as a character string |
| `include_assigned_obj` | A Boolean flag. If TRUE (default), it also returns the name of the assigned object. If FALSE, it drops the name of the assigned object. The FALSE flag is useful for when checking if the we wish to match the assigned object from a previous call with the actual argument in this call. |
| `count` | For `detect_growing`, this is a logical value that indicates if the number of "grow" expressions should be counted and returned, or if just a logical value should be returned. |
| `within_for` | If TRUE, only expresssions within a for loop are included. |
| `sub_fn` | (For count_fn_in_fn), the function to count (to look for within fn_name). |
| `arg` | The argument to check for within fn_name (as a character string). |

## Details

These are examples of functions that be called on a list of TreeHarp objects, which we refer to as a forestharp object. Such objects are not formally defined yet, but can be created using `rmd_to_forestharp` or using `join_treeharps`.

## Value

On their own, each of these functions should return a scalar or a 1-dimensional array. When called with `fapply`, the scalar numerical values can be combined (by taking the sum, any other provided combiner function).

The ultimate idea is that fapply should return a single feature for each rmd file that it is called upon.

## Functions

- `count_self_fn()`: Counts the number of self-defined functions.

  This helper counts the number of self-defined functions. It excludes lambda functions. It returns an integer scalar.

  As long as the function `function` was called and assigned, it will be counted.
- `count_lam_fn()`: Counts the number of anonymous functions.

  Counts the number of anonymous functions, typically used in sapply, etc. It returns an integer scalar. As long as the function `function` was called but *not* assigned, it will be counted here.
- `count_fn_call()`: Counts the number of function calls that match a pattern.

  This helper counts the number of function calls that match a pattern. It returns a count, i.e. an integer vector of length 1.

  If `pkg_name` is provided instead of `pattern`, then this function counts the number of function calls from that package.
- `extract_fn_call()`: Extracts function calls as a string.

  Extracts the function calls that match a pattern. It returns a character vector. Remember to set `combine = FALSE` when calling `fapply` with it.

- `extract_formal_args()`: Extracts function formal arguments called.
  Extracts the function *formal* arguments from functions with a given name. The name must match the function name exactly. This returns a character vector or NULL, if no formal arguments are used.
- `extract_assigned_objects()`: Extracts names of assigned objects
  Extracts the names of assigned objects. This was written to assist in detecting missed opportunities to use the pipe operator.
- `extract_actual_args()`: Extracts actual argument names
  Extracts the actual arguments from an expression, not the formal arguments. It only returns syntactic literals. It should be improved to return the actual arguments for a specified function so that something similar to `extract_assigned_objects` could be returned.
- `count_num_lines_for_loop()`: Counts number of lines in a for loop
  Extracts the actual arguments from an expression, not the formal First, this function checks if the expression contains a for loop. If it does, the innermost `for` loop is identified. Finally, the number of expressions below it is counted and returned.
- `detect_growing()`: Detects if a vector is being grown.
  It detects if there is an expression of form: x <- c(x, new_val). This is generally bad programming practice
- `detect_for_in_fn_def()`: Detects if a for loop is present within a function
  It detects if a for loop is present within a function definition.
- `count_fn_in_fn()`: Count use of a function within another.
  It counts the number of times a function is used within another.
- `detect_fn_call_in_for()`: Detect for loop to call a function
  Checks if a function has been called within a for loop.
- `extract_self_fn()`: Extract names of functions defined by user.
  Extracts names of user-defined functions. They may not all look nice, because sum functions may be anonymous functions. This function needs to be improved.
- `detect_fn_arg()`: Was a function called with a particular argument?
  Checks if a function was called with a particular argument, which could be the formal or actual one. The immediate child of the function call node is checked.
- `detect_nested_for()`: Was a nested "for" loop called anywhere within the code?
  Checks if a nested for-loop was called anywhere within the code. This returns a logical scalar for each TreeHarp object given.

## Examples

```
# Dummy trees
th1 <- TreeHarp(quote(X <- rnorm(10, mean=0.9, sd=4)), TRUE)
th2 <- TreeHarp(quote(Y <- rbeta(10, shape1=3, shape2=5)), TRUE)
th3 <- TreeHarp(quote(fn1 <- function(x) x + 2), TRUE)
th4 <- TreeHarp(quote(df1 <- mutate(df1, new_col=2*old_col)), TRUE)

# Run helpers
count_self_fn(th3)
count_fn_call(th4, pkg_name="dplyr")
count_fn_call(th1, pattern="^r.*")
```

## `generate_all_subtrees` *Generate all subtrees from a tree.*

### Description

This routines generates all subtrees rooted at the root node for a particular tree.

### Usage

```
generate_all_subtrees(th)
```

### Arguments

| | |
|---|---|
| `th` | An object of class TreeHarp. |

### Value

A 0-1 matrix with n rows and m columns. n is the number of sub-trees rooted at the root node of th. m is the number of nodes in this given tree. The leading column will be a 1 for all the rows.

### References

*Listing and counting subtrees of a tree*, F Ruskey, *SIAM Journal on Computing*, 1981

### See Also

`get_next_subtree`

### Examples

```
th1 <- TreeHarp(list(a=c(2,3), b=NULL, c=NULL))
generate_all_subtrees(th1)
```

## `generate_thumbnails` *Generate a html of thumbnails*

### Description

Generate a html of thumbnails

### Usage

```
generate_thumbnails(out_dir, html_fname, html_title, anonymise = FALSE)
```

### Arguments

| | |
|---|---|
| `out_dir` | The directory in which student html files and the figures are kept. |
| `html_fname` | The name of the master html file which will contain all thumbnails. This file will be created in `out_dir`. |
| `html_title` | The title tag of the master html page. This will be displayed on top of the output html page. |
| `anonymise` | If TRUE, the original filenames will be replaced with inocuous numbers. If FALSE, the original filenames will be retained. |

### Details

After running `render_one` on a set of R/Rmd files in a directory, this function helps to consolidate them for review.

The output folder contains all the generated html files, images and a log file. This function will extract the images from each html file and display them as thumbnails on a new html page, with links to all individual files.

### Value

The function returns nothing, but it should create a html page of thumbnails of all the images that students plotted, along with links to their individual pages.

---

## `get_adj_list` *Generic for Getting Adjacency List*

---

### Description

The generic method definition for getting adjacency list from a TreeHarp object.

### Usage

```
get_adj_list(x, ...)

## S4 method for signature 'TreeHarp'
get_adj_list(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | An object of class TreeHarp. |
| `...` | Unused arguments, for now. |

### Value

The adjacency list for a TreeHarp object.

### Methods (by class)

- `get_adj_list(TreeHarp)`: A getter.
  Allows user to extract the adjacency list of a treeharp object.

---

## `get_child_ids` *Generic for Getting Child Node Ids*

---

### Description

The generic method definition for getting child node ids.

### Usage

```
get_child_ids(x, node_num)

## S4 method for signature 'TreeHarp'
get_child_ids(x, node_num)

## S4 method for signature 'list'
get_child_ids(x, node_num)
```

### Arguments

| | |
|---|---|
| `x` | An object of class TreeHarp. |
| `node_num` | An integer, length 1. This the node whose children we are after. If the specified node is a leaf, the NULL is returned. |

### Value

An integer vector, indicating the children node ids.

### Methods (by class)

- `get_child_ids(TreeHarp)`: Obtain child nodes.
  Allows user to extract the child nodes from a specified node from TreeHarp object.
- `get_child_ids(list)`: Obtain child nodes.
  Allows user to extract the child nodes from a specified node from an adjacency list.

### See Also

`get_parent_id`

## `get_child_ids2` *Get the children node ids*

### Description

This function retrieves the child node ids of a given node from an adjacency list of a tree.

### Usage

```
get_child_ids2(adj_list, at_node)
```

### Arguments

| | |
|---|---|
| `adj_list` | The adjacency list of the tree. |
| `at_node` | The node whose children should be extracted. |

### Details

Remember that the list has to be for a tree, not a general graph. Please see other help pages for more specifications.

This is a low-level function, used within the S4 class TreeHarp. It is not generally meant for use by the user.

### Value

A vector of integers specifying the children of that particular node. If the node is a leaf, it returns NULL.

## `get_levels` *Obtains the node levels from a tree.*

### Description

This function obtains the node levels from a tree.

### Usage

```
get_levels(adj_list)
```

### Arguments

| | |
|---|---|
| `adj_list` | The adjacency list of the tree. |

### Details

This function is used to check if the specification of the tree is in BFS order. If that is indeed the case, the levels of each node should be sorted.

This function is not exported for the general user.

### Value

It returns a vector of integers. The length of this vector will be the number of nodes in the tree. The root is at level 1, the next is at level 2, and so on.

---

## `get_libraries` *Extracts the Packages Used in An Rmd File.*

---

### Description

The input filename could correspond to an R script or an Rmd file.

### Usage

```
get_libraries(fname)
```

### Arguments

| | |
|---|---|
| `fname` | The Rmd filename or R script. |

### Details

The file is assumed to be either an R script or an Rmd file. If it is found to be an Rmd file using extract_chunks, it is purl-ed before libraries are extracted. If it is found to be NOT an Rmd, it is assumed to be an R script and nothing is done to process it.

The file is not parsed, so even text files will work with this function.

### Value

A character vector containing the packages used within the Rmd document.

## `get_next_depth_id` *Get the id and depth of a child node.*

### Description

From the parent's depth and the last labelled node, we obtain the node id and depth of a child.

### Usage

```
get_next_depth_id(parent_node_id, env_ni)
```

### Arguments

| | |
|---|---|
| `parent_node_id` | The id of the parent node we are considering. |
| `env_ni` | An environment object, possibly containing a data frame with columns id, name, call_status, arg_type and depth. |

### Details

This is for internal use. It may be removed from user-view soon!

### Value

A list containing the id and depth of the next node.

## `get_next_subtree` *Generate the next sub-tree.*

### Description

This generates the next sub-tree in the enumeration list.

### Usage

```
get_next_subtree(obj, char_arr)
```

### Arguments

| | |
|---|---|
| `obj` | An object of class TreeHarp. |
| `char_arr` | A vector of 1's and 0's indicating which nodes to keep. The vector should have length equal to the number of nodes in obj. |

### Details

Need to reference the paper. This generates the next sub-tree, rooted at the root node of this tree. It will generate singletons on it's own. It has to be used within a loop to do that.

## Value

A vector of 1's and 0's, which denotes the next sub-tree in the list.

## See Also

generate_all_subtrees

## Examples

```
th1 <- TreeHarp(list(a=c(2,3), b=NULL, c=NULL))
get_next_subtree(th1, c(1,0,0))
get_next_subtree(th1, c(1,1,0))
```

---

`get_node_types` *Generic for Getting Node Types*

---

## Description

The generic method definition for getting node types from a TreeHarp object.

## Usage

```
get_node_types(x, ...)

## S4 method for signature 'TreeHarp'
get_node_types(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class TreeHarp. |
| `...` | Unused arguments, for now. |

## Value

A data frame containing the node types for a TreeHarp object. If the slot is empty, NA is returned.

## Methods (by class)

- `get_node_types(TreeHarp)`: A getter.
  Allows user to extract the node types of a treeharp object.

## `get_parent_call_id` *Get Node Id of Parent Call*

### Description

Get the node id of the parent call for a given node.

### Usage

```
get_parent_call_id(x, node_id)
```

### Arguments

| | |
|---|---|
| `x` | A TreeHarp object. |
| `node_id` | The id of the node whose parent call is to be found. An integer value. |

### Details

When we need to go up the parse tree to obtain the function that called this node, we use this function. It is similar to `get_parent_id`, except that that function only returns the immediate parent.

It is not useful to call this function when the TreeHarp object is not constructed from a language object.

Perhaps this function is necessary only because of the way language objects are represented by the autoharp: formal arguments are included in the tree representation. When we wish to find the calling function, we have to walk up the branches till we reach a function call.

### Value

An integer corresponding to the node id of the calling function.

### See Also

`get_parent_id`

### Examples

```
ex3 <- quote(x <- f(y = g(3, 4), z=1L))
t1 <- TreeHarp(ex3, TRUE)

# get the function that calls g:
get_parent_call_id(t1, 6)
#contrast with this:
get_parent_id(t1, 6)
```

## get_parent_id *Generic for Getting Parent Node Id.*

### Description

The generic method definition for getting parent node id.

### Usage

```
get_parent_id(x, node_num)

## S4 method for signature 'TreeHarp'
get_parent_id(x, node_num)

## S4 method for signature 'list'
get_parent_id(x, node_num)
```

### Arguments

| | |
|---|---|
| `x` | An object of class TreeHarp or an adjacency list. |
| `node_num` | An integer, length 1. This the node whose parent we are after. If node_num is equal to 1, then NULL is returned because that should be the root node. |

### Value

An integer, indicating the parent node.

### Methods (by class)

- `get_parent_id(TreeHarp)`: Obtain parent node id.
  Extracts parent id of a node from a TreeHarp object.
- `get_parent_id(list)`: Obtain parent node id.
  Extracts parent id of a node from an adjacency list object.

### See Also

`get_child_ids`

## `get_parent_id2` *Get the parent node id*

### Description

This function retrieves the parent node id of a given node from an adjacency list of a tree.

### Usage

```
get_parent_id2(adj_list, at_node)
```

### Arguments

| | |
|---|---|
| `adj_list` | The adjacency list of the tree. |
| `at_node` | The node whose parent should be extracted. |

### Details

Remember that the list has to be for a tree, not a general graph. Please see other help pages for more specifications.

This is a low-level function, used within the S4 class TreeHarp. It is not generally meant for use by the user.

If there are nodes that have more than one parent, then a warning is issued.

### Value

A integer of length 1 should be returned for all nodes except the root. For the latter, the function will return NULL.

## `get_recursive_index` *Obtain an index to extract out a sub-call*

### Description

Obtains an index that can be used to extract a sub-call from a language object.

### Usage

```
get_recursive_index(th, node_id)
```

### Arguments

| | |
|---|---|
| `th` | A TreeHarp object. |
| `node_id` | An integer corresponding to a call within the parse tree (not a literal, symbol or a formal argument). |

## Value

A vector of indices, that can be used (together with "[[") to obtain a sub-call

## Examples

```
ex3 <- quote(x <- f(y = g(3, 4), z=1L))
t1 <- TreeHarp(ex3, TRUE)
rec_index <- get_recursive_index(t1, 6)
ex3[[rec_index]]
ex3[[get_recursive_index(t1, 3)]]
```

---

`is_connected` *Checks if a graph is connected.*

---

## Description

A tree is a graph that is connected but does not have any cycles. This function checks if a provided adjacency list is connected.

## Usage

```
is_connected(adj_list, root = 1)
```

## Arguments

| | |
|---|---|
| `adj_list` | The adjacency list of the tree. |
| `root` | The root node to start checking from. This defaults to the first node in the adjacency list. |

## Details

This function is used as one of the validity checks within the definition of the TreeHarp class. It is a low-level function, not really meant for the general user of the package. Hence it is not exported.

The nodes are traversed in a BFS order. The function could actually be combined with is_cyclic_r, but it is kept separate for modularity reasons.

An alternative was to convert the list to an adjacency matrix and check for a row and column of zeros.

## Value

The function returns a TRUE if the graph is connected and FALSE otherwise.

## `is_cyclic_r` *Checks if a graph contains any cycles.*

### Description

A tree is a graph that is connected but does not have any cycles. This function checks if a provided adjacency matrix contains cycles.

### Usage

```
is_cyclic_r(adj_mat, node_v, parent_node = -1, visited_env)
```

### Arguments

| | |
|---|---|
| `adj_mat` | A symmetric matrix of 1's and 0's, with 1 in entry (i,j) representing an edge between the two vertices. |
| `node_v` | The node to begin searching for cycles from. An integer. |
| `parent_node` | The parent node of node_v. Also an integer. Use -1 if you are starting from node 1. This is in fact the default. |
| `visited_env` | An environment containing a logical vector indicating which nodes have already been visited. The vector has to be named "visited". See the details. The function works by traversing all the nodes, in a BFS order. If it finds a node has a parent that has already been visited, it concludes that there is a cycle. The function is recursive, and has to update the vector of visisted nodes within each call. Hence the visited vector is stored in an environment that is passed along. It will return an error if no such environment is provided. It is a very specific input that the function requires, and this is another reason that this function is not exported. This function is used within the validity checks for the S4 class. It is not exported for the user. |

### Value

A logical value indicating if the graph contains cycles.

## `is_subtree_rooted_at` *Checks if a tree is rooted at a node of another tree.*

### Description

This function checks if a given tree is a sub-tree of another tree at a particular node.

### Usage

```
is_subtree_rooted_at(x, y, at_node)
```

### Arguments

| | |
|---|---|
| `x` | An object of class TreeHarp. |
| `y` | An object of class TreeHarp. |
| `at_node` | An integer, corresponding to a node in object y. The sub-tree of y, rooted at at_node, is compared to x. |

### Details

Here's how it works: The sub-tree of y, rooted at at_node is first extracted. The tree x is then compared to this. If x is a sub-tree of it, then this function returns FALSE. Otherwise it returns TRUE.

### Value

A logical value indicating if x is a sub-tree of y, rooted at at_node.

### Examples

```
thb1 <- TreeHarp(list(b=2, d=NULL))
tha1 <- TreeHarp(list(a=c(2,3), b=4, c = NULL, d=NULL))
is_subtree_rooted_at(thb1, tha1, 1) # FALSE
is_subtree_rooted_at(thb1, tha1, 2) # TRUE
```

---

## `jaccard_treeharp` *Computes Jaccard Index*

---

### Description

Computes the Jaccard index between two trees.

### Usage

```
jaccard_treeharp(th1, th2, weighted = FALSE)
```

### Arguments

| | |
|---|---|
| `th1` | A TreeHarp object. |
| `th2` | A TreeHarp object. |
| `weighted` | A logical value, indicating if the weighted Jaccard similarity should be computed. |

### Details

The unweighted form is just the cardinality of the intersection of the two sets of tokens, divided by the union of the two sets.

The weighted form is described on the WIkipedia page: https://en.wikipedia.org/wiki/Jaccard_index#Weighted_Jaccard_simi

### Value

A real number between 0 and 1.

---

## `join_treeharps` *Root a list of trees.*

---

### Description

Given a list of trees, this will root them.

### Usage

```
join_treeharps(...)
```

### Arguments

| | |
|---|---|
| `...` | A list of Treeharp objects. |

### Details

This function combines TreeHarp objects into a single TreeHarp. The function will root all of them at a node called "script", which is neither a function call nor an argument nor a symbol. The BFS ordering is then updated.

Objects that are not of class TreeHarp will be dropped from the list before the rooting takes place.

### Value

A TreeHarp object

---

## `K2` *Compute tree similarity*

---

### Description

Compute tree similarity

### Usage

```
K2(t1, t2, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `t1` | A TreeHarp object. |
| `t2` | A TreeHarp object. |
| `verbose` | A logical value, indicating if the output should be verbose. |

### Details

As far as possible, this function tries to do things recursively. It sets up a n x m matrix and fills up as much as it can. Then it uses recursive relationships to fill in the rest. When it cannot, it uses `generate_all_subtrees` to generate and count common subtrees.

### Value

An integer, that counts the number of sub-trees in common between the two trees. Please see the reference papers for more information.

### References

1. *Convolution kernels for natural language*, M Collins and N Duffy, *Advances in neural information processing systems*, 2002.
2. *Convolution kernels on discrete structures*, D Haussler, *Technical report, Department of Computer Science, UC Santa Cruz*, 1999.

### Examples

```
tree1 <- TreeHarp(quote(x <- 1), TRUE)
tree2 <- TreeHarp(quote(y <- 1), TRUE)
K2(tree1, tree2, TRUE)
```

---

## `keep_branches` *Keep only branches specified by node numbers*

---

### Description

Retains only specific branches, that are identified by their node numbers.

### Usage

```
keep_branches(th, branch_nodes, include_lower = TRUE)
```

### Arguments

| | |
|---|---|
| `th` | A TreeHarp object. |
| `branch_nodes` | An integer vector, specifying the nodes to keep. |
| `include_lower` | A logical value - whether or not the lower branches should also be kept. |

### Value

A TreeHarp object.

## Examples

```
ex1 <- quote(x <- f(y, g(5)))
th1 <- TreeHarp(ex1, TRUE)
keep_branches(th1, 3)
keep_branches(th1, 3, include_lower = FALSE)
keep_branches(th1, c(2,3), FALSE)
keep_branches(th1, c(3, 4), FALSE)
```

---

`lang_2_tree` *Convert language object to tree.*

---

## Description

A recursive function for converting a language object to treeharp.

## Usage

```
lang_2_tree(lang_obj, node_id, ni_env)
```

## Arguments

| | |
|---|---|
| `lang_obj` | A language object. |
| `node_id` | The calling node to this language object. This should only be greater than 0 if the `ni_env` already contains a partial adjacency list and corresponding node information. This will happen when this function is called recursively. |
| `ni_env` | An environment to store the adjacency list and node information. |

## Details

This function is used by TreeHarp constructors. It should not have to be called by a user. It works by bulding up an adjacency list and node node information data frame within the supplied environment.

## Value

Nothing

## Examples

```
e1 <- new.env()
lang_2_tree(quote(X <- 1), 0, e1)
e1$adj_list
e1$node_info
```

## `log_summary` *Generate a dataframe from the log file.*

### Description

Generate a dataframe from the log file.

### Usage

```
log_summary(log_file)
```

### Arguments

| | |
|---|---|
| `log_file` | The name of the log file generated from `render_one`. |

### Details

This provides a table view of the log file, which is updated in a more natural format by simply concatenating new updates. The output of this function makes it easier to group entries by filename, time, or status, or even error message.

The output table does not contain correctness output. It only contains the columns name, timestamp, status (SUCCESS/FAIL), error message, number of libraries used and number of libraries installed.

### Value

The function returns a dataframe summarising the details in the log file.

### See Also

`render_one`

## `matrix_2_adj_list` *Convert adjacency matrix to a list.*

### Description

Converts a binary matrix that represents a tree into an adjacency list.

### Usage

```
matrix_2_adj_list(mat)
```

### Arguments

| | |
|---|---|
| `mat` | A symmetric matrix of 1's and 0's, with 1 in entry (i,j) representing an edge between the two vertices. |

### Details

Remember that the list that is finally output is for a tree, not a general graph. Please see other help pages for more specifications.

The input matrix should be BFS ordered. The adjacency list only notes the child node(s) of a particular node. If a matrix denotes multiple parents, it will not be picked up.

This is a low-level function, used within the S4 class TreeHarp. It is not generally meant for use by the user.

### Value

The adjacency list of the tree.

---

## `only_actual_args_differ` *Checks if two expressions differ only in terms of actual arguments*

---

### Description

Checks if two expressions differ only in terms of actual arguments

### Usage

```
only_actual_args_differ(th1, th2)
```

### Arguments

| | |
|---|---|
| `th1` | A TreeHarp object. |
| `th2` | A TreeHarp object. |

### Details

From the node types, only calls and formal arguments are retained. If these are identical, then a 1 is returned. Otherwise 0 is returned.

### Value

Either 1 or 0. 1 means that the two expressions differ only in their actual arguments.

### Examples

```
ex1 <- TreeHarp(quote(X <- rnorm(10, mean=0.9, sd=4)), TRUE)
ex2 <- TreeHarp(quote(Y <- rnorm(20, mean=9, sd=4)), TRUE)
only_actual_args_differ(ex1, ex2)
```

## path_to_root *Extract a path from node to root.*

### Description

Identifies the nodes on the path from a node up to the root of a TreeHarp object.

### Usage

```
path_to_root(th, node_num)
```

### Arguments

| | |
|---|---|
| `th` | A TreeHarp object. |
| `node_num` | A node number to start tracking upwards from. |

### Details

This function allows the user to identify the branch from a node up to the root of a tree.

### Value

A vector of 1's and 0's that can be used to carve out the branch alone, using `carve_subtree`.

### Examples

```
ex1 <- quote(x <- f(y, g(5)))
th1 <- TreeHarp(ex1, TRUE)
path_to_root(th1, 5)
```

## pipe_can_improve_this *Identify if use of pipe operator can improve a code section*

### Description

Identify if use of pipe operator can improve a code section

### Usage

```
pipe_can_improve_this(fname, window_len = 2)
```

### Arguments

| | |
|---|---|
| `fname` | A filename - either a Rmd/qmd or R script. |
| `window_len` | A window length to analyse. |

## Details

A rolling window approach is used here. For each expression, the assigned object is extracted (if any). Subsequent lines are checked to see if this object appears as an actual argument.

Window length two means that only one more line is checked.

## Value

A numeric vector corresponding to start of the window to be reviewed. If no lines are found, NULL is returned.

---

## `plot,TreeHarp-method` *TreeHarp Plotting TreeHarp Objects*

---

### Description

A plot method for visualising treeharp objects.

### Usage

```
## S4 method for signature 'TreeHarp'
plot(x, y, ...)
```

### Arguments

| | |
|---|---|
| `x` | An object of class TreeHarp. |
| `y` | Unused. |
| `...` | Additional arguments passed on to plot.igraph(). |

### Details

The treeharp object is converted to an igraph object before it is plotted.

### Value

Returns NULL, invisibly.

## populate_soln_env *Returns solution environment and test code from template.*

### Description

Generates objects for checking solution correctness.

### Usage

```
populate_soln_env(
  soln_fname,
  pattern,
  knit_root_dir,
  render_only = FALSE,
  output = NULL,
  dummy = FALSE
)
```

### Arguments

| | |
|---|---|
| `soln_fname` | An rmd file containing the checks to be run on the student solution. |
| `pattern` | The pattern that identifies which chunks in the solution are are testing chunks. If this argument is missing, the default pattern used is "test". |
| `knit_root_dir` | The root directory to use for knitting the rmd file. This argument is optional. If it is missing, it uses the root directory in knitr::opts_knit$get('root.dir'). |
| `render_only` | A logical value. If this is TRUE, then the solution is run, rendered and returned. Otherwise the rendered html is deleted. |
| `output` | The path to the knitted solution md file. This is usually deleted immediately, but sometimes we may want to keep it. This argument is passed on to `knit`, so please refer to that page for the warnings about setting this argument when figures are involved. |
| `dummy` | A logical value. If TRUE, then a dummy solution object is created The environment will be empty, and the temporary test script will only have one line in it: 'ah_dummy <- TRUE'. This solution object can be used to render a solution file without any correctness checks. Of course, one could just knit it directly, but this is still useful when testing things out. |

### Details

Test code should be written in a chunk that generates scalars from student objects.

The solution file has to be an Rmd file (not an R script), because it relies on the autoharp.obj and autoharp.scalars knitr hooks being present.

In addition, if it is required that a solution object is to be tested against the analogous object within the student environment, these objects should be listed within the autoharp.objs option of a code chunk. These objects will be copied with the "." prefix.

Here is an overview of how the function works:

1. Knit the solution file to generate the solution (or "correct") objects.
2. Rename these with the "." prefix in the solution environment object.
3. Extract the lines of test code into a temporary R script.
4. Wrap those chunks that contain autoharp.scalars hook with tryCatch.
5. Return the solution environment and path to the R test script.

Typically, the next step is to call `check_correctness`.

### Value

If render_only is FALSE, a list containing 2 components: the environment populated by the solution rmd and the path to an R script containing the test code.

If render_only is TRUE, then the output list consists of the aforementioned environment, and the path to the rendered solution file (html). This option is useful for debugging the solution file.

### See Also

`check_correctness`, `render_one`

---

## `prune_depth` *Prune a tree up to a specified depth.*

---

### Description

Prunes a tree up to a depth specified by a set of node names.

### Usage

```
prune_depth(th, names_to_keep)
```

### Arguments

| | |
|---|---|
| `th` | A TreeHarp object. |
| `names_to_keep` | The node names to keep in the pruned tree. |

### Details

This is a seldom used function. It works in this way. Given a set of node names, it identifies the node with the greatest depth in that set. The function then returns the sub-tree, that contains all the nodes with a depth smaller than or equal to that depth. If the node types slot is not NA, then that data frame is filtered and returned too.

Take a look at the examples for a clearer picture.

### Value

An object of class TreeHarp.

## See Also

carve_subtree, path_to_root, carve_mst

## Examples

```
ex1 <- quote(x <- f(y, g(5)))
th1 <- TreeHarp(ex1, TRUE)
s1 <- prune_depth(th1, c("f", "y"))
s2 <- prune_depth(th1, c("f", "z")) # node not present!
plot(s1)
plot(s2)
```

---

## `rbind_to_nodes_info` *Update node information.*

---

### Description

Updates the node information regarding an R expression.

### Usage

```
rbind_to_nodes_info(id, name, call_status, formal_arg, depth, env_ni)
```

### Arguments

| | |
|---|---|
| `id` | The id of the node to be added. This should be an integer of length 1. |
| `name` | The name of the node. |
| `call_status` | Is the language object a call or a symbol/literal? This should a logical value. |
| `formal_arg` | Is the language object a formal argument or not? This should be a logical value. |
| `depth` | An integer indicating the depth of this language object in the parse tree. |
| `env_ni` | An environment object, possibly containing a data frame with columns id, name, call_status, formal_arg and depth. |

### Details

This is for internal use. It may be removed from user-view soon!

### Value

TRUE is returned invisibly.

## render_one *Run a single Rmd file through autoharp.*

### Description

Renders the specified file, and collates run time, static and correctness checks.

### Usage

```
render_one(
  rmd_name,
  out_dir,
  knit_root_dir,
  log_name,
  soln_stuff,
  max_time_per_run = 120,
  permission_to_install = FALSE
)
```

### Arguments

| | |
|---|---|
| `rmd_name` | The path to the file to be rendered and checked. |
| `out_dir` | The directory to store all the html output, md output, and figures. |
| `knit_root_dir` | The working directory while knitting the file. |
| `log_name` | A character string, denoting the log file name. It defaults to "render_one.log". If this file is already present in the directory, this function will append to it. |
| `soln_stuff` | This is a list, with components env, test_fname, and tt_list. This object is the output of `populate_soln_env`. Set this to be NA if you wish to skip correctness checks, and only do rendering. |
| `max_time_per_run` | The maximum time to wait before aborting the rendering of a particular file. |
| `permission_to_install` | If TRUE, then the function will try to install any packages needed. By default, this is FALSE. |

### Details

The log file contains a record of the libraries used by the student, and if any new libraries needed to be installed. The status will be one of SUCCESS, FAIL or UNKNOWN.

### Value

A data frame with one row for each file in the input directory.

### See Also

`populate_soln_env`, `check_correctness`

## `render_prechecks` *Conducts checks before rendering file*

### Description

This routine conducts two checks before rendering the file: presence of View function, and presence of system() calls. The former is a nuisance, while the latter is possibly nefarious.

### Usage

```
render_prechecks(fname, in_place = TRUE, new_name, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `fname` | The name of the Rmd/qmd file to check. |
| `in_place` | If TRUE, the original file will be overwritten in place. |
| `new_name` | If in_place is FALSE, this should be the full path to the new file to be written (including directory location). |
| `verbose` | If TRUE, debugging messages will be printed. |

### Value

Either TRUE or FALSE. If TRUE, all View() lines have been commented, and there are no system calls. If FALSE, it means there are system calls (to be screened further).

## `replace_sp_chars_filename` *Replace Special Characters in File Name*

### Description

Replaces special characters in the name of an R or Rmd script.

### Usage

```
replace_sp_chars_filename(dir_name, return_df = TRUE)
```

### Arguments

| | |
|---|---|
| `dir_name` | A character string, referring to the directory of Rmd files whose names should be replaced. |
| `return_df` | A logical value, indicating if the old and new names should be returned (in a tibble). |

### Details

If a filename contains one of the following special characters (ignore the quotes here): `"[ <>()|\:&;#?*']"`, the knit function will replace them with underscores. Hence the filenames in the autoharp input directory and the output directory will not match, even allowing for the change in file extension. This will cause problems when we try to run `render_one` again on the same input directory.

This function renames the files in the input directory by replacing all special characters there.

The NUS LMS (LumiNUS) introduces parenthesized names or numbers in order to make filenames unique, so this function is necessary for NUS instructors.

### Value

A tibble containing the old and new names.

---

## `reset_path` *Reset search path of current R session*

---

### Description

This function is used to detach packages that have been added by a student script.

### Usage

```
reset_path(old_path)
```

### Arguments

| | |
|---|---|
| `old_path` | A character vector of package namespaces. This is usually the output of `search`, run before an R script or Rmd file is rendered, which could cause the search path to change. |

### Details

When a student script is rendered using `render_one`, new packages might be added to the search path. These may conflict with the instructors' search path order, or with subsequent runs of `render_one` on students. Hence there is a need to reset the search path before this is done.

This function does not unload namespaces. It only detaches them from the search path. For a difference between the two, please see Hadley's page.

### Value

There is no object returned. This function is called for it's side- effect of altering the search path.

### Examples

```
opath <- search()
# Load a package
reset_path(opath)
```

## rmd_to_forestharp *Convert to TreeHarp objects*

### Description

Reads in an Rmd file or an R script and converts it to a list of TreeHarp objects.

### Usage

```
rmd_to_forestharp(fname, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `fname` | The filename that is to be read in. |
| `verbose` | If TRUE, then messages will be printed to enable debugging. dropped. |

### Details

If an Rmd or qmd file is supplied as input, it is first converted to an R script before `get_source_expressions` is used to try to parse the expressions. These expressions are then matched to the original file to retrieve line numbers. Unlike previous versions of this function, line numbers will always be returned.

Expressions that could not be parsed will be returned as NA.

Line numbers are extracted using `get_source_expressions` from the lintr package.

### Value

There are three possible return outcomes: (1) NA, indicating that `purl` was unable to extract the R expressions from the Rmd/qmd file. (2) A list with two components, named `forest` and `line_nums`, each being NA. (3) A list with two components, named as in (2), containing the TreeHarp objects and a vector of line numbers.

If there are syntax errors in the file, the expressions may not be parsed correctly. The `verbose` option will reflect this.

### See Also

`fapply`, `extract_chunks`, `extract_chunks`, `get_source_expressions`

## `rmd_to_token_count` *Count tokens in R/Rmd*

### Description

Count the individual tokens. Part of the NLP analysis process.

### Usage

```
rmd_to_token_count(fname, include_actuals = TRUE)
```

### Arguments

| | |
|---|---|
| `fname` | The Rmd or R file name. |
| `include_actuals` | Whether actual arguments/literals should be included. If this is FALSE, then only calls and formal arguments will be used in the count. |

### Value

A tibble. The tibble will contain a the frequency count for all tokens present in the student script.

## `subtree_at` *Extract a sub-tree.*

### Description

Extracts a sub-tree rooted at a particular node.

### Usage

```
subtree_at(obj, at_node, preserve_call = FALSE)
```

### Arguments

| | |
|---|---|
| `obj` | An object of class TreeHarp |
| `at_node` | The root of the new sub-tree. An integer, not a label, that corresponds to BFS indexing of the tree. |
| `preserve_call` | A logical value that indicates if a sub-call should be extracted. This might be slower, but it allows you to evaluate it later. |

### Details

This is meant for internal use, so the nodeTypes slot is silently dropped, unless preserve_call is set to TRUE

### Value

An object of class TreeHarp.

### Examples

```
th3 <- list(a= c(2L,3L,4L), b=NULL, c=c(5L, 6L), d=7L, e=NULL, f=NULL, g=NULL)
subtree_at(TreeHarp(th3), 3)
st <- subtree_at(TreeHarp(th3), 4)
plot(st)
```

## to_BFS *Function to rearrage nodes in BFS*

### Description

Function to rearrage nodes in BFS

### Usage

```
to_BFS(adj_list, node_info)
```

### Arguments

| | |
|---|---|
| `adj_list` | The output of lang_2_tree. |
| `node_info` | The output of lang_2_tree. |

### Details

This function is for an internal TreeHarp constructor use. It is not exported.

### Value

An adjacency list and nodes info data frame in BFS order.

## TreeHarp-class *An R expression as a tree.*

### Description

This class is used to represent a *single* R expression as a tree.

### Usage

```
TreeHarp(lang_obj, quote_arg, ...)

TreeHarp(lang_obj, quote_arg, ...)

## S4 method for signature 'logical'
TreeHarp(lang_obj, quote_arg, ...)

## S4 method for signature 'missing'
TreeHarp(lang_obj, quote_arg, ...)

## S4 method for signature 'TreeHarp'
length(x)

## S4 method for signature 'TreeHarp'
show(object)

## S4 method for signature 'TreeHarp'
names(x)
```

### Arguments

| | |
|---|---|
| `lang_obj` | This should be an adjacency list for a tree (not a graph), or the adjacency matrix of a tree, or the expression to be parsed. If it is a list, only child nodes should be indicated (see the examples). |
| `quote_arg` | If this argument is missing or FALSE, then the class of `lang_obj` will be evaluated, and, if it is either a list or matrix, the TreeHarp object will be returned. If this argument is TRUE, the `lang_obj` argument will be quoted and a parse tree for the expression will be computed and used as the tree. |
| `...` | Unused at the moment. |
| `x` | A Treeharp object. |
| `object` | A TreeHarp object. |

### Details

The following validity checks are conducted on the object:

1. Is the graph connected? If no, the object is invalid.

2. Are there cycles? If yes, the object is invalid.
3. Are the nodes labelled in a BFS ordering? If not, the object is not valid.

## Value

Constructors return an object of class TreeHarp.

`length`: An integer of length 1.

`print`: Returns NULL. It prints a string representation of a TreeHarp object.

`names`: A character vector with length equal to the number of nodes.

## Methods (by generic)

- `TreeHarp(logical)`: A constructor for TreeHarp.
  Converts either adjacency list or matrix into a TreeHarp object.
- `TreeHarp(missing)`: A constructor for TreeHarp.
  Converts language object into a TreeHarp object.
- `length(TreeHarp)`: To get the length of a tree.
  The length of the tree refers to the number of nodes in the tree.
- `show(TreeHarp)`: To print a tree representation.
  A string representation of a TreeHarp object.
- `names(TreeHarp)`: To get tree labels
  This function returns the node labels of the tree.

## Slots

`adjList` The adjacency list of the tree. The list must be named. The nodes should be labelled in Breadth-First Order. The first component must be the root of the tree. Leaves of the tree should be NULL elements.

`nodeTypes` A data frame describing the type of node. The columns in the data frame will be derived from the expression used to instantiate the object. The column names will be id (node id), name, call_status, formal_arg and depth. This slot can be left missing (i.e., populated with NA). This latter feature is useful when we just wish to test something out.

This slot is only populated automatically when an R expression is provided as `lang_obj` and `quote_arg` is TRUE.

`repr` A string representation of the tree. This will be printed when the show method of TreeHarp is called.

`call` The language object that was used to construct the tree (if it was). If the object was constructed from a list/matrix, this will be NA.

## Examples

```
l1 <- list(a=c(2,3), b=NULL, c=NULL)
# directly using new()
treeharp1 <- new("TreeHarp", adjList = l1, nodeTypes = NA)
```

```
# using one of the constructor methods (for lists)
treeharp2 <- TreeHarp(l1)

# using the constructor for matrices.
m1 <- matrix(0L, 3, 3)
dimnames(m1) <- list(letters[1:3], letters[1:3])
m1[1, ] <- c(0, 1L, 1L)
m1[, 1] <- c(0, 1L, 1L)
treeharp3 <- TreeHarp(m1)

# Supplying a language object to get the same tree (with nodeTypes
# populated)
ex1 <- quote(a(b,c))
TreeHarp(ex1, TRUE)
```

## tree_sim *Compute tree similarity*

### Description

Computes similarity between two trees (non-recursively)

### Usage

```
tree_sim(t1, t2, norm = FALSE, ...)
```

### Arguments

| | |
|---|---|
| t1 | A TreeHarp object |
| t2 | Anothe TreeHarp object. |
| norm | A logical value, indicating if the kernel function should be normalised, to account for different tree lengths. |
| ... | Unused arguments, reserved for mcmapply |

### Value

A numerical value between 0 and 1 (if normed).

## update_adj_list *Update adjacency list.*

### Description

Updates the adjacency list for an R expression parse tree.

### Usage

```
update_adj_list(
  update_type = c("new_node", "add_child"),
  node_id,
  node_name,
  child_node,
  env_ni
)
```

### Arguments

| | |
|---|---|
| update_type | This should be either "new_node" or "add_child". If it is a new node, an empty list component is added. If it is add child, then child_node should be provided too. |
| node_id | An integer. |
| node_name | The name of the new node to be added. This must be provided if the update_type is "new_node". |
| child_node | An integer. |
| env_ni | An environment object, possibly containing an adjacency list that will later be used to construct a TreeHarp object. |

### Details

This is for internal use. It may be removed from user-view soon!

### Value

An invisible TRUE is returned.

# Index

adj_list_2_matrix, 3
apply_can_improve_this, 4
as.matrix, 4
autoharp, 5
autoharp-package (autoharp), 5

carve_mst, 6, *42*
carve_subtree, 7, *38*, *42*
check_correctness, 7, *41*, *43*
check_rmd, 8
check_runtime, 9
clean_dir, 10
copy_e2e, 10
count_fn_call (forestharp-helpers), 17
count_fn_in_fn (forestharp-helpers), 17
count_lam_fn (forestharp-helpers), 17
count_lints_all, 11
count_lints_one, 12
count_num_lines_for_loop (forestharp-helpers), 17
count_self_fn (forestharp-helpers), 17

detect_fn_arg (forestharp-helpers), 17
detect_fn_call_in_for (forestharp-helpers), 17
detect_for_in_fn_def (forestharp-helpers), 17
detect_growing (forestharp-helpers), 17
detect_nested_for (forestharp-helpers), 17

env_size, 12
extract_actual_args (forestharp-helpers), 17
extract_assigned_objects (forestharp-helpers), 17
extract_chunks, 13, *24*, *46*
extract_fn_call (forestharp-helpers), 17
extract_formal_args (forestharp-helpers), 17
extract_non_chunks, 14
extract_section_text, 14
extract_self_fn (forestharp-helpers), 17

fapply, 15, *17*, *18*, *46*
find_branch_num, 16
forestharp-helpers, *15*, 17

generate_all_subtrees, 20, *26*, *34*
generate_thumbnails, 20
get_adj_list, 21
get_adj_list,TreeHarp-method (get_adj_list), 21
get_child_ids, 22, *28*
get_child_ids,list-method (get_child_ids), 22
get_child_ids,TreeHarp-method (get_child_ids), 22
get_child_ids2, 23
get_levels, 23
get_libraries, *9*, *18*, 24
get_next_depth_id, 25
get_next_subtree, *20*, 25
get_node_types, 26
get_node_types,TreeHarp-method (get_node_types), 26
get_parent_call_id, 27
get_parent_id, *22*, *27*, 28
get_parent_id,list-method (get_parent_id), 28
get_parent_id,TreeHarp-method (get_parent_id), 28
get_parent_id2, 29
get_recursive_index, 29
get_source_expressions, *46*

is_connected, 30
is_cyclic_r, *30*, 31
is_subtree_rooted_at, 31

jaccard_treeharp, 32

join_treeharps, *18*, 33

K2, 33
keep_branches, 34
knit, *40*, *45*

lang_2_tree, 35
length,TreeHarp-method
        *(*TreeHarp-class*)*, 49
log_summary, 36

matrix_2_adj_list, 36

names,TreeHarp-method *(*TreeHarp-class*)*,
        49

only_actual_args_differ, 37

path_to_root, 38, *42*
pipe_can_improve_this, 38
plot,TreeHarp-method, 39
populate_soln_env, *5*, *8*, 40, *43*
prune_depth, 41
purl, *46*

rbind_to_nodes_info, 42
render_one, *5*, *8*, *9*, *13*, *21*, *36*, *41*, 43, *45*
render_prechecks, 44
replace_sp_chars_filename, 44
reset_path, 45
rmd_to_forestharp, *18*, 46
rmd_to_token_count, 47

search, *45*
show,TreeHarp-method *(*TreeHarp-class*)*,
        49
subtree_at, 47

to_BFS, 48
tree_sim, 51
TreeHarp, *5*
TreeHarp *(*TreeHarp-class*)*, 49
TreeHarp,logical-method
        *(*TreeHarp-class*)*, 49
TreeHarp,missing-method
        *(*TreeHarp-class*)*, 49
TreeHarp-class, 49

update_adj_list, 52