

# Package ‘tailr’

December 15, 2018

**Version** 0.1.2

**Title** Automatic Tail Recursion Optimisation

**Description** Implements meta-programming functions for automatically translating recursive functions into looping functions or trampolines.

**License** GPL-3

**Encoding** UTF-8

**Language** en-GB

**LazyData** true

**ByteCompile** true

**Depends** R (>= 3.2)

**Imports** rlang (>= 0.2.0), glue

**Suggests** covr, testthat, microbenchmark, compiler, pmatch (>= 0.1.2)

**RoxygenNote** 6.0.1

**URL** <https://github.com/mailund/tailr>

**BugReports** <https://github.com/mailund/tailr/issues>

**NeedsCompilation** no

**Author** Thomas Mailund [aut, cre]

**Maintainer** Thomas Mailund <mailund@birc.au.dk>

**Repository** CRAN

**Date/Publication** 2018-04-27 07:54:53 UTC

## R topics documented:

build_transformed_function . . . . .	2
can_call_be_transformed . . . . .	2
can_loop_transform_body . . . . .	3
can_transform_rec . . . . .	4
handle_recursive_returns . . . . .	4
handle_recursive_returns_call . . . . .	5
loop_transform . . . . .	5
make_returns_explicit . . . . .	6
make_returns_explicit_call . . . . .	6
returns_to_escapes . . . . .	7

returns_to_escapes_call . . . . .	7
simplify_nested_blocks . . . . .	8
simplify_returns . . . . .	8
simplify_returns_call . . . . .	9
translate_recursive_call . . . . .	9
user_transform . . . . .	10
user_transform_rec . . . . .	10
<b>Index</b>	<b>11</b>

---

build\_transformed\_function

*Construct the expression for a transformed function body.*

---

## Description

This is where the loop-transformation is done. This function translates the body of a recursive function into a looping function.

## Usage

```
build_transformed_function(fun_expr, info)
```

## Arguments

fun_expr	The original function body.
info	Information passed along the transformations.

## Value

The body of the transformed function.

---

can\_call\_be\_transformed

*Tests if a call object can be transformed.*

---

## Description

Tests if a call object can be transformed.

## Usage

```
can_call_be_transformed(call_name, call_arguments, fun_name, fun_call_allowed,
cc)
```

**Arguments**

call_name	Name (function) of the call.
call_arguments	The call's arguments
fun_name	The name of the recursive function we want to transform
fun_call_allowed	Whether a recursive call is allowed at this point
cc	Current continuation to abort if a transformation is not possible

**Value**

TRUE, if the expression can be transformed. Invokes cc otherwise.

---

can_loop_transform_body	<i>Tests if a function, provided by its name, can be transformed.</i>
-------------------------	---

---

**Description**

This function analyses a recursive function to check if we can transform it into a loop or trampoline version with [transform](#). Since this function needs to handle recursive functions, it needs to know the name of its input function, so this must be provided as a bare symbol.

**Usage**

```
can_loop_transform_body(fun_name, fun_body, fun, env)

can_loop_transform_(fun)

can_loop_transform(fun)
```

**Arguments**

fun_name	Name of the recursive function.
fun_body	The user-transformed function body.
fun	The function to check. Must be provided by its (bare symbol) name.
env	Environment used to look up variables used in fun_body.

**Functions**

- can\_loop\_transform\_body: This version expects fun\_body to be both tested and user-transformed.
- can\_loop\_transform\_: This version expects fun to be quosure.
- can\_loop\_transform: This version quotes fun itself.

**Examples**

```
factorial <- function(n)
  if (n <= 1) 1 else n * factorial(n - 1)
factorial_acc <- function(n, acc = 1)
  if (n <= 1) acc else factorial_acc(n - 1, n * acc)

can_loop_transform(factorial)      # FALSE -- and prints a warning
can_loop_transform(factorial_acc)  # TRUE

can_loop_transform_(rlang::quo(factorial))      # FALSE -- and prints a warning
can_loop_transform_(rlang::quo(factorial_acc))  # TRUE
```

---

can_transform_rec	<i>Recursive call for testing if an expression can be transformed into a looping tail-recursion.</i>
-------------------	--

---

**Description**

Recursive call for testing if an expression can be transformed into a looping tail-recursion.

**Usage**

```
can_transform_rec(expr, fun_name, fun_call_allowed, cc)
```

**Arguments**

expr	The expression to test
fun_name	The name of the recursive function we want to transform
fun_call_allowed	Whether a recursive call is allowed at this point
cc	Current continuation, used to escape if the expression cannot be transformed.

**Value**

TRUE, if the expression can be transformed. Invokes cc otherwise.

---

handle_recursive_returns	<i>Handle the actual recursive calls</i>
--------------------------	--

---

**Description**

Handle the actual recursive calls

**Usage**

```
handle_recursive_returns(expr, info)
```

**Arguments**

expr	An expression to transform
info	Information passed along the transformations.

**Value**

A modified expression.

---

handle_recursive_returns_call	<i>Handles the actual recursive returns</i>
-------------------------------	---

---

**Description**

This function dispatches on a call object to set the context of recursive expression modifications.

**Usage**

```
handle_recursive_returns_call(call_expr, info)
```

**Arguments**

call_expr	The call to modify.
info	Information passed along with transformations.

**Value**

A modified expression.

---

loop_transform	<i>Transform a function from recursive to looping.</i>
----------------	--

---

**Description**

Since this function needs to handle recursive functions, it needs to know the name of its input function, so this must be provided as a bare symbol.

**Usage**

```
loop_transform(fun, byte_compile = TRUE)
```

**Arguments**

fun	The function to transform. Must be provided as a bare name.
byte_compile	Flag specifying whether to compile the function after transformation.

---

`make_returns_explicit` *Make exit points into explicit calls to return.*

---

### Description

Make exit points into explicit calls to return.

### Usage

```
make_returns_explicit(expr, in_function_parameter, info)
```

### Arguments

<code>expr</code>	An expression to transform
<code>in_function_parameter</code>	Is the expression part of a parameter to a function call?
<code>info</code>	Information passed along the transformations.

### Value

A modified expression.

---

`make_returns_explicit_call`  
*Make exit points into explicit calls to return.*

---

### Description

This function dispatches on a call object to set the context of recursive expression modifications.

### Usage

```
make_returns_explicit_call(call_expr, in_function_parameter, info)
```

### Arguments

<code>call_expr</code>	The call to modify.
<code>in_function_parameter</code>	Is the expression part of a parameter to a function call?
<code>info</code>	Information passed along with transformations.

### Value

A modified expression.

---

returns_to_escapes	<i>Make calls to return into calls to escapes.</i>
--------------------	--

---

**Description**

Make calls to return into calls to escapes.

**Usage**

```
returns_to_escapes(expr, info)
```

**Arguments**

expr	An expression to transform
info	Information passed along the transformations.

**Value**

A modified expression.

---

returns_to_escapes_call	<i>Make calls to return into calls to escapes.</i>
-------------------------	--

---

**Description**

This function dispatches on a call object to set the context of recursive expression modifications.

**Usage**

```
returns_to_escapes_call(call_expr, info)
```

**Arguments**

call_expr	The call to modify.
info	Information passed along with transformations.

**Value**

A modified expression.

---

`simplify_nested_blocks`*Simplify nested code-blocks.*

---

**Description**

If a call is `{` and has a single expression inside it, replace it with that expression.

**Usage**

```
simplify_nested_blocks(expr)
```

**Arguments**

<code>expr</code>	The expression to rewrite
-------------------	---------------------------

**Value**

The new expression

---

`simplify_returns`*Remove `return(return(...))` expressions*

---

**Description**

Remove `return(return(...))` expressions

**Usage**

```
simplify_returns(expr, info)
```

**Arguments**

<code>expr</code>	An expression to transform
<code>info</code>	Information passed along the transformations.

**Value**

A modified expression.



---

`simplify_returns_call` *Removes `return(return(...))` cases.*

---

### Description

This function dispatches on a call object to set the context of recursive expression modifications.

### Usage

```
simplify_returns_call(call_expr, info)
```

### Arguments

<code>call_expr</code>	The call to modify.
<code>info</code>	Information passed along with transformations.

### Value

A modified expression.

---

`translate_recursive_call`  
*Translate a `return(<recursive-function-call>)` expressions into a block that assigns the parameters to local variables and call next.*

---

### Description

Translate a `return(<recursive-function-call>)` expressions into a block that assigns the parameters to local variables and call next.

### Usage

```
translate_recursive_call(recursive_call, info)
```

### Arguments

<code>recursive_call</code>	The call object where we get the parameters
<code>info</code>	Information passed along to the transformations.

### Value

The rewritten expression

---

user_transform	<i>Apply user transformations depths-first.</i>
----------------	---

---

**Description**

Apply user transformations depths-first.

**Usage**

```
user_transform(expr, fun = expr, env = rlang::caller_env())
```

**Arguments**

expr	The expression to transform – typically a function body.
fun	The actual function to transform.
env	The environment where functions can be found.

**Value**

Rewritten expression

---

user_transform_rec	<i>Apply user transformations depths-first.</i>
--------------------	---

---

**Description**

The difference between this function and `user_transform` is that the this function does not perform type checks before calling recursively while `user_transform` does.

**Usage**

```
user_transform_rec(fun, expr, env)
```

**Arguments**

fun	The actual function to transform.
expr	The expression to transform – typically a function body.
env	The environment where functions can be found.

**Value**

Rewritten expression

# Index

build\_transformed\_function, [2](#)

can\_call\_be\_transformed, [2](#)

can\_loop\_transform  
    (can\_loop\_transform\_body), [3](#)

can\_loop\_transform\_  
    (can\_loop\_transform\_body), [3](#)

can\_loop\_transform\_body, [3](#)

can\_transform\_rec, [4](#)

handle\_recursive\_returns, [4](#)

handle\_recursive\_returns\_call, [5](#)

loop\_transform, [5](#)

make\_returns\_explicit, [6](#)

make\_returns\_explicit\_call, [6](#)

returns\_to\_escapes, [7](#)

returns\_to\_escapes\_call, [7](#)

simplify\_nested\_blocks, [8](#)

simplify\_returns, [8](#)

simplify\_returns\_call, [9](#)

transform, [3](#)

translate\_recursive\_call, [9](#)

user\_transform, [10](#), [10](#)

user\_transform\_rec, [10](#)