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`

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

Tests if a function, provided by its name, can be transformed.

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

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 *Handle the actual recursive calls*

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 *Handles the actual recursive returns*

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 *Transform a function from recursive to looping.*

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

expr	An expression to transform
in_function_parameter	Is the expression part of a parameter to a function call?
info	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

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

Value

A modified expression.

`returns_to_escapes` *Make calls to return into calls to escapes.*

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` *Make calls to return into calls to escapes.*

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

`expr`	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

`expr`	An expression to transform
`info`	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

call_expr	The call to modify.
info	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

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

Value

The rewritten expression

user_transform *Apply user transformations depths-first.*

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 *Apply user transformations depths-first.*

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