CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER'S THESIS

Title:	Improvements of the RIR bytecode toolchain
Student:	Bc. Jan Ječmen
Supervisor:	Ing. Petr Máj
Study Programme:	Informatics
Study Branch:	System Programming
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2018/19

Instructions

Familiarize yourself with the R language, its bytecode compiler, and interpreter architecture. Familiarize yourself with RIR, an alternative bytecode format, compiler, and interpreter for the language. The R bytecode compiler assumes certain invariants (such as built-in meaning of control flow statements and certain operators) about the code to make the compiled code faster. Analyze similar assumptions that are used by RIR and extend RIR to use assumptions made by GNU-R as well. Identify and implement improvements to the RIR (compiler, bytecode format, and interpreter). Discuss your results.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague March 1, 2017

Czech Technical University in Prague

Faculty of Information Technology

Department of Theoretical Computer Science

Master's thesis

Improvements of the RIR bytecode toolchain

Bc. Jan Ječmen

Supervisor: Ing. Petr Máj

April 30, 2017

Acknowledgements

[[acknowledgements]] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Declaration

Citation of this thesis

JEČMEN, Jan. *Improvements of the RIR bytecode toolchain.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017. Available also from WWW: ⟨`https://github.com/JanJecmen/dip`⟩.

Abstract

[[abstract english]] This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Keywords **[[keywords en]]**

Abstrakt

[[abstract czech]] And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Klíčová slova [[keywords cz]]

Contents

List of Figures

List of Listings

List of Tables

Introduction

[[write intro; cite: https://www.tiobe.com/tiobe-index/ http://pypl.github.io/PYPL.html]] After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This thesis is structured in the following way:

The chapter *About GNU R* gives a short introduction to GNU R and discusses its features. It also goes under the hood and describes the inner workings of its interpreter and bytecode compiler.

The chapter *About RIR* introduces an alternative bytecode compiler for the R language, talks about the motivation behind it, its architecture and design choices, the differences to the original and its shortcomings.

The chapter *Improvements* describes in depth the changes that were done to RIR in this thesis.[[?]]

The chapter *Evaluation* discusses how the performance of RIR changed after the changes done in this thesis. It describes how the measurements were done and how the performance compares to GNU R.[[?]]

The results of this thesis are discussed in *Conclusion*, as well as the direction of future efforts regarding RIR.

CHAPTER 1

About GNU R

GNU R[1] is a programming language used mainly for statistical computations. It is an open-source dialect of S, an older statistical language created in 1976 by John Chambers at Bell Laboratories. R has been around from 1993 and was designed by Ross Ihaka and Robert Gentleman, both recognised statisticians. It is a part of the GNU software family and is still actively developed by the R Core Team today. It is a popular alternative to the other major implementation of the S language, S-PLUS, which is a commercial version shipped by TIBCO Software Inc. **[[cite]]**

R comes with a software environment built around it, which allows for easily manipulating data, carrying out computations and producing quality graphical outputs such as plots and figures. Although at heart R is used via a command line interface, there are also more user-friendly graphical IDEs available. One of the most widely used is RStudio[2] that provides, for example, syntax highlighting and quick access to documentation through a web-like browser. This, together with R's readable syntax and a vast collection of extension packages available through The Comprehensive R Archive Network (CRAN) makes it possible for new users to step in and start working quickly.

[1]Homepage: `https://www.r-project.org/`
[2]Homepage: `https://www.rstudio.com/`

1.1 Language features of R

[[cite: http://r.cs.purdue.edu/pub/ecoop12.pdf]]

[[cite: http://adv-r.had.co.nz/]]

R is, as far as programming languages go, very interesting and has some quite unusual semantic features. It is an interpreted language, and is dynamically typed and garbage collected. It supports multiple programming paradigms: users can use procedural imperative style, but at the same time R provides an object system (more than one, in fact!) for object oriented programming, and is heavily influenced by functional programming languages, notably Scheme (a dialect of Lisp).

Functions are, in accordance with functional languages, first-class values, so they can be passed around as call arguments, returned as results of function calls and created dynamically at runtime. R uses lexical scoping (which it adopted from Scheme) and R functions are closures that capture their enclosing environment at creation time. Arguments are passed by value (although a variant of reference counting is implemented, so that deep copies are only created as needed, e.g., when an object is modified).

Interestingly, everything that happens in R is in fact a function call. This goes as far as arithmetic operators being just syntactic sugar for function calls, as can be seen in listing 1.1[3]. In this spirit, even assigning into a variable, evaluating a block of code inside curly brackets or grouping expressions with parentheses translate to calling the respective functions.

All actual arguments to a function are lazy evaluated by default. When applying a closure, parameters are wrapped in promises. A promise is simply an object that contains the unevaluated expression and an environment in which the expression should be evaluated. Promises are only evaluated when the value is actually needed (which is called forcing the promise). Also, once the promise is forced, it remembers the result, so that subsequent uses of the value do not evaluate the expression again.

[3]As a side note, backticks are used it R to denote symbols (i.e., names). Because some symbols have special syntactic meaning (like the `` `+` `` being an infix binary operator), they can cause a syntax error if they appear unquoted in a place where the parser does not expect them.

```
> typeof(`+`)
[1] "builtin"
> `+`
function (e1, e2)  .Primitive("+")
> `+`(1, 2)
[1] 3
> 1 + 2
[1] 3
```

Listing 1.1: Arithmetic operators are function calls in R

This is demonstrated in listing 1.2, where the function in question only evaluates its first argument and does not touch the second. For the first call, the block of code in the curly brackets is never executed and so the side-effect of printing a greeting does not happen. In the second call, the arguments are swapped, and the side-effect can be observed in the output. It is possible to pass a block of code as a function argument, since the `` `{` `` is bound to a function that sequentially evaluates all its arguments and returns as its result the value of the last expression (or `NULL` if no expressions are passed in).

```
> f <- function(a, b) a
> f(1, {cat("Hello\n"); 2})
[1] 1
> f({cat("Hello\n"); 2}, 1)
Hello
[1] 2
```

Listing 1.2: Promise lazy evaluation

These features highlight the functional approach of R by minimizing side-effects. However, R supports assignment[4] which enables programmers to change a function's local state by modifying its bindings and thus the imperative programming style. Also, the superassignment operator `` `<<-` `` makes it possible to change non-local bindings (as shown in listing 1.3) and thus brings the side-effects back into play.

The basic data type in R is an atomic vector. Vectors are collections of homogeneous values (i.e., a given vector can only hold objects of one particular

[4]An interesting quirk is R's left to right assignment: the code `1 -> x` assigns 1 to x and is equivalent to `x <- 1`.

```
> x <- 1
> f <- function() {
+     x <- 2  # local
+     x <<- 3  # lookup in the enclosing environment
+     x  # local
+ }
> x
[1] 1
> f()
[1] 2
> x
[1] 3
```

Listing 1.3: Superassignment

type), that preserve the order of their elements. R also provides a list type which is heterogeneous. Higher-dimensional types such as matrices and data frames, as well as objects, are built from vectors and lists under the hood. In R there are no scalar types, as scalar values, such as individual numbers and strings, are considered to be vectors of length one.

Atomic vectors can have one of these six types: logical, integer, double, character, complex and raw. Since R targets data analysis, a special “not available” value `NA` is provided for these. Because all values in a vector must be of the same type, R performs coercion when an attempt is made to combine vectors of different types. In listing 1.4, combining a numeric vector with a character vector results in a character vector, and summing a logical vector coerces to integers (`TRUE` becoming `1` and `FALSE` becoming `0`).

Despite its inspiration in functional world, R does not optimize tail recursion, which is the standard approach in functional languages since they typically use recursion in the place of iterative loops. Instead, R encourages vectorized operations. Hence, most of R builtin functionality works element-wise with vectors, while recycling the elements as needed (e.g., when adding vectors of different lengths). This is demonstrated in listing 1.5, where R even issues a warning about recycling.

In R, every object can have arbitrary attributes associated with its data. Attributes are basically a hidden map that assigns names to values. Some of the most important attributes are names (a character vector that assigns names to

```
> x <- c(1, 2, 3)
> y <- c("a", "b", "c")
> typeof(x)
[1] "double"
> typeof(y)
[1] "character"
> c(x, y)
[1] "1" "2" "3" "a" "b" "c"
> typeof(c(x, y))
[1] "character"
>
> sum(c(TRUE, TRUE, FALSE, TRUE))
[1] 3
```

Listing 1.4: Coercion to the most flexible type

```
> numeric(10)
 [1] 0 0 0 0 0 0 0 0 0 0
> 1:3
[1] 1 2 3
> numeric(10) + 1:3
 [1] 1 2 3 1 2 3 1 2 3 1
Warning message:
In numeric(10) + 1:3 :
  longer object length is not a multiple of shorter object length
```

Listing 1.5: Recycling shorter vector

elements), dimensions (a vector specifying the dimensions and thus effectively distinguishing vectors from matrices and arrays) and class (for implementing one of R's object systems). Attributes are used a lot in R for many purposes and extensions as they provide a way of encoding arbitrary additional metadata for objects. As an example, in listing **??** a vector of 4 elements is created, then changed into a 2x2 matrix and finally changed back to vector with named elements.

[[data frames?]]

[[objects]]

[[subsetting, subassignment]]

[[ellipsis, named arguments, missing]]

```
> x <- 1:4
> x
[1] 1 2 3 4
> attr(x, "dim") <- c(2, 2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> attr(x, "dim") <- c(4)
> attr(x, "names") <- c("a", "b", "c", "d")
> x
a b c d
1 2 3 4
```

Listing 1.6: Object attributes

1.1.1 AST interpreter

In its core R uses a classic architecture for an interpreted language. After initialization, the user enters R's read-eval-print loop (REPL), that lets them type in expressions and have R evaluate them. First, a reader, or parser, waits for the user input and reads it line by line. If, at the end of line, it has read a syntactically complete expression, it passes it to an evaluator (otherwise it waits for more input). After the evaluator returns the evaluated expression, a printer is invoked that displays the result (with some exceptions, such as assignment that sets its result to be invisible). Then the reader is again invoked and the process repeats.

Every object in R is internally represented by a C structure called SEXPREC[5] (actually, R passes the objects around as pointers to this structure, which are called SEXP). This structure contains a header with metadata about the object, such as its type, reference counter or infromation for garbage collector, and then a union of other structures that represent different types of R internal objects. Some of these types are listed it table 1.1.

[5]The name refers to S-expressions, or symbolic expressions, as known from Lisp, although the classical linked lists built from dotted pairs are mostly used internally, and vectors are implemented as C arrays for efficiency reasons.

Table 1.1: Some common types of internal R objects

Type	Usage
NILSXP	the singleton NULL object
SYMSXP	symbols (or names)
LISTSXP	lists of dotted pairs
CLOSXP	closures
ENVSXP	environments
PROMSXP	promises
LANGSXP	language constructs (typically closure application)
SPECIALSXP	special forms (typically control flow)
BUILTINSXP	builtin non-special forms (e.g., arithmetic operators)
INTSXP	integer vectors
REALSXP	real vectors
STRSXP	string vectors
BCODESXP	object compiled to bytecode

The parser, when it scans the stream of input characters, checks that it is syntactically correct and at the same time builds a tree structure that represents the parsed expression. This tree is called the abstract syntax tree (AST) and its nodes are all SEXPs. An example AST is shown in the listing 1.7[6]. In the listing, parentheses mean function call (i.e., LANGSXP node), the first child being the callee (typically a SYMSXP, i.e., a name that is bound to a function) and the rest its arguments.

The evaluator is a recursive function that gets as its input two SEXP objects, one representing the AST of the expression that is to be evaluated, and the other the environment in which to evaluate the expression. The evaluator walks the given AST and based on the type of nodes it encounters, performs some action. The result is then returned to be processed by the caller of eval.

[6] **[[about pryr]]**

```
> pryr::ast(x <- (y + 3) * f(z))
\- ()
  \- `<-
  \- `x
  \- ()
    \- `*
    \- ()
      \- `(
      \- ()
        \- `+
        \- `y
        \-  3
    \- ()
      \- `f
      \- `z
```

Listing 1.7: AST of a simple expression

Some nodes are self-evaluating, meaning that no action needs to be performed and the node itself is the result. These are for example the NULL object, the atomic vectors or the environments.

If the eval function sees a symbol node, a lookup for its binding is perfomed in the provided environment. If it is not found there, because of lexical scoping, the parent environment is searched, and so on, until either the binding is found or an empty environment is reached (the empty environment serves as a sentinel parent of all environment chains and does not have a parent itself).

One other prominent type of nodes is LANGSXP. R has internally three types of functions, called special, builtin and closures (or user-defined functions). These have different behavior when they are applied, and the eval function handles that.

Special funcions are the core language constructs, such as control flow (conditionals and loops). They take their arguments unevaluated in a list and evaluate them as needed while running. This is necessary for example for the if[[verb]] statement, because, since R has side-effects, only one of the conditional branches must be evaluated to preserve the correct semantics.

Builtins, on the other hand, are known to evaluate all their arguments, so it is not necessary to create promises from their arguments. Instead, a list of evaluated arguments is created and passed to the builtin function. Examples of builtin functions are arithmetic operators or the colon operator for generating integer sequences.

The last group are closures. Closures are user-defined functions written in R, and they adhere to the lazy evaluation semantics. All arguments to a closure are therefore allocated as promises: the expressions are bundled together with the enclosing environment.

As opposed to specials and builtins, which, being C routines, are called directly after preparing their arguments, closures need the interpreter to do some additional work. First, the actual unevaluated arguments have to be matched to the formal arguments of the closure. Then, a new environment has to be created and filled with the matched pairs or arguments. Only after this can the body of the closure be evaluated in the new environment. Also, a longjump target is set here to catch any explicit return calls from within the body.

Finally, the dispatch to the bytecode interpreter for objects compiled to bytecode is also found in the eval function.

1.1.2 BC compiler and interpreter

In an attempt to make R faster, a special internal representation for R code was developed, and a compiler from R to this bytecode was added in a package called **[[verb]]**compiler. This also required some minor changes to the original AST interpreter, namely adding a new SEXP type for the compiled objects, called BCODESXP, and handling of bytecode objects in the evaluator. A new evaluator was also added for interpreting the bytecode which is invoked by the AST version when it needs to evaluate a compiled object.

The compiler was written by Luke Tierney, and added as a standard package to R in 2011 (version 2.13.0**[[cite: https://stat.ethz.ch/pipermail/r-announce/2011/000538.html]]**). However, it was not used by default until version 3.4.0, released in late April 2017[7].**[[cite https://stat.ethz.ch/pipermail/r-announce/2017/000612.html]]**

[[cite compiler pdf]] The BC compiler itself is implemented in R, and walks the abstract syntax tree of an expression being compiled in a similar manner

[7]Although packages were compiled when installed**[[since when? all?]]**

that the AST interpreter does (but, of course, it does so at the R level by using introspection). However, instead of evaluating the code as it traverses the tree, it produces a code object. The code is then executed by a virtual machine runtime system.

The virtual machine that executes the R bytecode is stack based. This means that at runtime a stack is used by the instructions to get their arguments and store their results. For example, the instruction that performs addition, expects its two operands at the top of the stack. When it is executed, it removes these two objects from the stack, adds them together, and puts the resulting object back on the top of the stack.

The bytecode objects produced by the compiler consist of two components. The first is an integer vector that encodes the code itself in the form of instruction opcodes interleaved with the instruction operands. The second is a general list that represents a constant pool. The compiler is designed such that each bytecode object has its own constant pool. In the constant pool, important objects are stored, such as the source for the compiled expression, small constant objects, or promises.

The compiler comes with a disassembler that makes it possible to inspect the bytecode, as is shown in listing 1.8. The object is printed as a list that starts with the `.Code` symbol, then follows the code vector with the opcodes decoded, and last comes the constant pool. The integers in the code that are not instructions represent arguments to the instructions (the first element is an exception, as it encodes the version of the BC stored in the given object).

```
> f <- compiler::cmpfun(function(n) n + 1)
> f
function(n) n + 1
<bytecode: 0x367ee40>
> compiler::disassemble(f)
list(.Code, list(8L, GETVAR.OP, 1L, LDCONST.OP, 2L, ADD.OP, 0L,
    RETURN.OP), list(n + 1, n, 1))
```

Listing 1.8: Disassembling a BC object

[[JIT in applyclosure]]

[[only now as default out of the box behaviour for base compiler written in R 120sth bc instructions interoperates with the normal eval, but for bytecode evaluation uses loop with switch (or threading) bytecode encoded in vector of ints, plus per object constant pool]]

1.2 Why is R hard to optimize

Firstly, R is not the fastest language out there. Of course, being an intepreted language, one cannot expect the performance of lagnuages like C that are compiled to native machine code. This is because during runtime, there is the inherent overhead of managing the virtual machine that executes a given program. In the case of AST interpreter, this overhead is [[TODO]]

R is a very dynamic language and gives the programmer a very high degree of freedom.

[[cite: https://cran.r-project.org/doc/manuals/R-lang.html section 6 and 2]]

[[dynamic, user can do anything non-standard evaluation introspection vectorized subsetting, subassignment delayed evaluation two different object systems r inferno some examples builtin, special, closures]]

[[first: dynamic nature, second: design desicions]] [[vanilla r uses standard repl and ast interpreter single threaded runtime type checking, coercion memory hungry, garbage collection, everything on stack, a lot of metadata and attributes everything is function call written in c, no jit by default, no native jit compiler]]

CHAPTER 2

About RIR

[[Introduce RIR]] RIR[8] is an alternative compiler for the R language. It comes with its own internal representation, an interpreter for its bytecode and an abstract interpretation framework which provides a way to easily implement static analyses on top of the RIR bytecode.

[[history: research project, northeastern? first appearence?]]

RIR acts as a drop-in replacement for the GNU R bytecode compiler. It requires a patched version of GNU R that makes some slight adjustments that allow the standard GNU R expression evaluator function to interface with the RIR bytecode compiler and interpreter. RIR is written in C and C++ and is compiled as a shared library that can be dynamically loaded by R.

[[write about rir bytecode]]

[[how is rir bc different]]

[[optimizations, ai framework...]]

[[architecture]]

[8]Homepage: `https://github.com/reactorlabs/rir`

2.1 Why is RIR slow

[[TODO]]This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

[[subsection about gnur compiler assumptions]]

CHAPTER 3

Improvements

In this chapter I will discuss in detail the changes made to RIR in an attempt to bring it up to speed with GNU R byte-compiled code.

...

3.1 Refactoring RIR interpreter

As it turned out, a lot of speedup could be gained by changing the RIR bytecode interpreter.

[[to compiler, to ir, to interpreter, use code snippets, describe microbenchmarks, theory (threaded code...)]] [[everywhere: motivation - how it helped in microbenchmarks, then how in real]] After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

```
f <- function() {
    i <- 10000000L
    while (i > 0) {
        i <- i - 1
    }
}
system.time(f())[[3]]  # jit everything
t <- c()
for (x in 1:15) t <- c(t, system.time(f())[[3]])
mean(t[5:15])  # only include measurments after warmup
```

Listing 3.1: **[[write listing caption]]** microbenchmark (run with jit enable 2)

[[relational operators, fast paths for logical args, unary plus minus not, loop contexts, bc cleanup, colon, superassing, inlining of instructions in main loop, threaded code, inline stack funcs, loops refactor, disable guardfuns]]

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really?

Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

CHAPTER 4

Evaluation

[[discussion of results, add figures]] [[discuss interesting point in measurements - naive nbody and threading etc.]]

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this

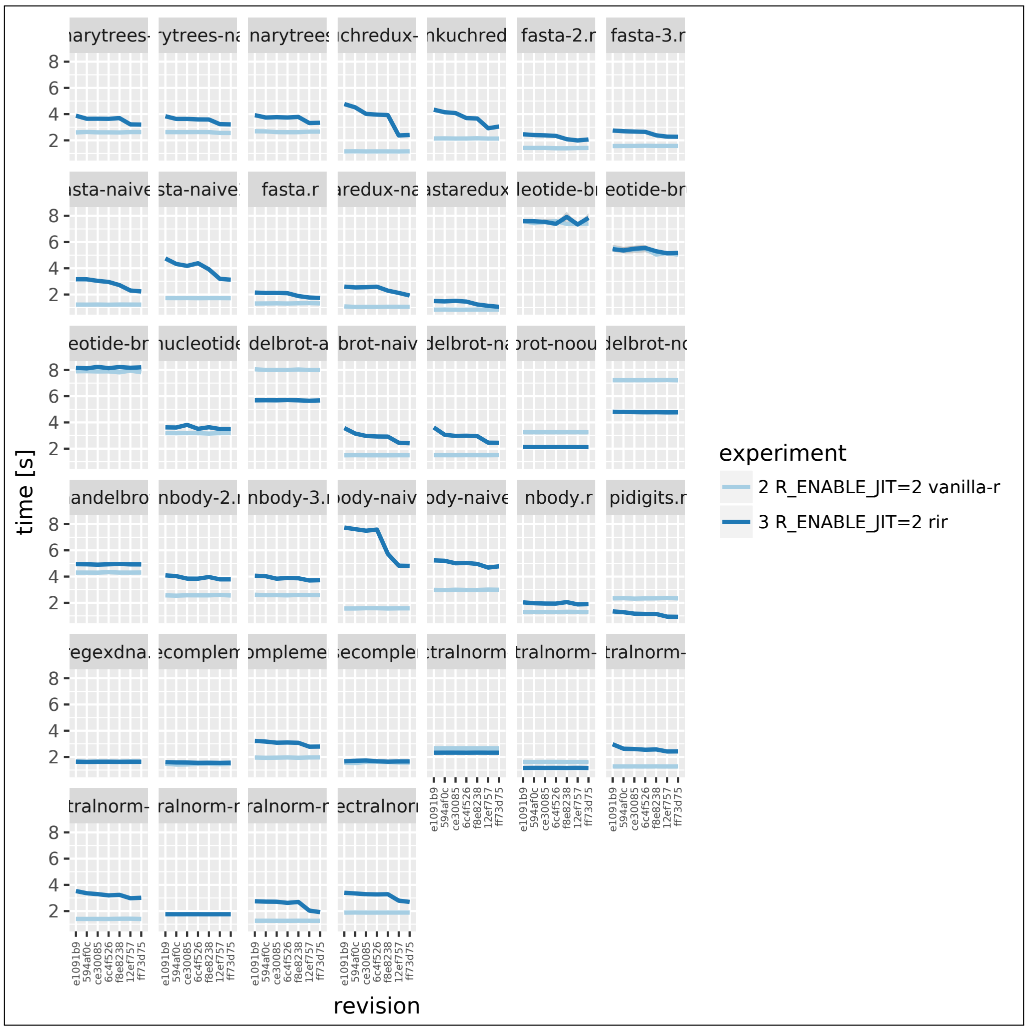

Figure 4.1: [[write title]]

text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

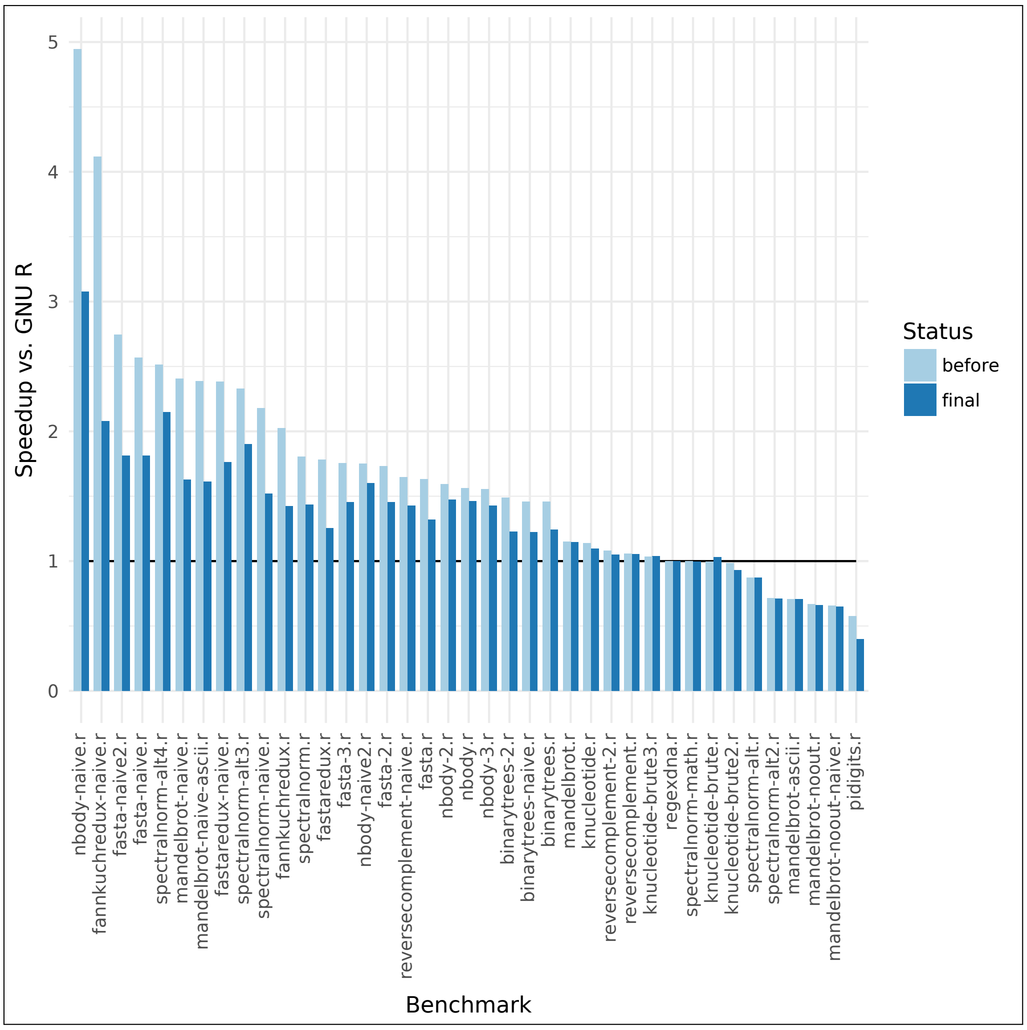

Figure 4.2: **[[write title]]**

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Conclusion

[[conclusion, future work, related work, fails - stoke etc.]] After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

APPENDIX A

Acronyms

API	Application programming interface
AST	Abstract syntax tree
BC	Bytecode
CLI	Command Line Interface
CRAN	The Comprehensive R Archive Network
GNU	GNU’s Not Unix!
REPL	Read-eval-print loop

APPENDIX B

Contents of the enclosed CD

[[contents of cd]] And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.