COMPILATION OF A TYPED R-LIKE LANGUAGE TO WEBASSEMBLY

Baljinnyam Bilguudei

Bachelor's thesis
Faculty of Information Technology
Czech Technical University in Prague
Department of Computer Science
Study program: Informatics
Specialisation: Computer Science
Supervisor: Pierre Donat-Bouillud, Ph.D.
December 28, 2025

Citation of this thesis: Baljinnyam Bilguudei. *Compilation of a typed R-like language to WebAssembly* . Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2026.

Chtěl bych poděkovat především sit amet, consectetuer adipiscing elit. Curabitur sagittis hendrerit ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Declaration

FILL IN ACCORDING TO THE INSTRUCTIONS. VYPLŇTE V SOULADU S POKYNY. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Curabitur sagittis hendrerit ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue. Donec ipsum massa, ullamcorper in, auctor et, scelerisque sed, est. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Pellentesque pretium lectus id turpis.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Curabitur sagittis hendrerit ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue. Donec ipsum massa, ullamcorper in, auctor et, scelerisque sed, est. In sem justo, commodo ut, suscipit at, pharetra vitae, orci. Pellentesque pretium lectus id turpis.

In Prague on December 28, 2025

Abstract

R is a widely-used dynamic language for statistical computing, but its dynamic nature prevents efficient ahead-of-time compilation. We present the design and implementation of a compiler for a statically-typed subset of R that targets WebAssembly. The defined language retains R's core characteristics, including first-class vector operations and lexical scoping, while introducing static typing to enable compilation. The compiler leverages the WebAssembly Garbage Collection proposal, which simplifies memory management and enables efficient representation of high-level language constructs. Source programs are transformed through parsing, type checking, and intermediate representation lowering to generate WebAssembly bytecode, supported by a runtime system providing vector operations and host environment interoperability. Evaluation on representative statistical programs demonstrates the viability of compiling R-like languages to WebAssembly using modern proposals.

Keywords WebAssembly, WASM GC, typed R, WASM bytecode, pass-based Compiler

Abstrakt

Fill in the abstract of this thesis in Czech. Lorem ipsum dolor sit amet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Klíčová slova enter, comma, separated, list, of, keywords, in, CZECH

Contents

List of Figures

List of Tables

List of Code listings

List of abbreviations

WASM	Web Assembly
WAT	Web Assembly Text
GC	Garbage Collector
IR	Intermediate Representation

Chapter 1
Introduction

The R programming language is ubiquitous for statistical computing and data analysis, offering powerful abstraction for data manipulation and visualization. Inspired by S-Language, the R language has been important part for industries working with statistics and been pioneer for introducing widespread tools currently used by the community such as DataFrame.

(probably reformat, explain how R is compiled and what web assembly is shortly) The thesis, wasmR, presents compilation of typed R-like language that compiles to WebAssembly directly. The reason choosing for R-like language, instead of plain R language is that dynamic languages are known to be notoriously hard to compile to a typed bytecode(reference needed). R programs are traditionally in the bucket of interpreted languages where types are only known runtime, and not in compile-time. However, compiling to typed bytecode requires full information of the types(why exactly, what are the reasons?), unless the architect is ready to face trade-off of writing generic functions that dispatches on runtime time.

An example below to show how the bytecode for generic functions would look like in arbitrary typed bytecode. How would we compile untyped code for?

1. The high level code

   ```
   c = a + b
   ```

2. Every value is a boxed value.

   ```
   Value {
     tag : TypeTag
     payload : union {
       int64
       float64
       pointer
       object_ref
   ```

```
  }
}

TypeTags :== INT | FLOAT | STRING | OBJECT | ...
```

3. Load variable and call the generic function (simplified instruction)

```
LOAD_LOCAL   a
LOAD_LOCAL   b
ADD_GENERIC
STORE_LOCAL  c
```

4. What ADD_GENERIC have to do?

```
b = pop()
a = pop()

if a.tag == INT and b.tag == INT:
    push(int_add(a, b))
elif a.tag == FLOAT and b.tag == FLOAT:
    push(float_add(a, b))
elif a.tag == STRING and b.tag == STRING:
    push(string_concat(a, b))
elif a.tag == OBJECT:
    call a.__add__(b)
else:
    runtime_type_error()
```

Remember, this is optimistic scenario. What happens when we have on LHS(left-hand side) an INT and on RHS(right-hand side) FLOAT? Moreover what if one of them is composite types? As one can see this dispatcher function for every operation combinated with every type will be a huge overhead.

1.1 Motivation

One main motivation for the thesis is WebAssembly as a compilation target. There's no work currently exactly mapping R to WebAssembly. Moreover, the recent WebAssembly GC proposal enables more convenient and efficient implementation of high-level language features including function references, closures, and composite data structures such as arrays and structs.

Chapter 2
Background

2.1 The R Programming Language

R is a domain-specific language designed for statistical computing and graphics, originally developed by Ross Ihaka and Robert Gentleman in the early 1990s as an open-source implementation of the **S** language. With its CRAN package manager providing more than 20000 packages, it's a widely programming language used by industries where experimenting with data is involved, such as data analytics, data mining and bio-informatics.

2.1.1 Key Characteristics

- **Dynamic typing:** Variables have no declared types; types are determined at runtime. This enables rapid prototyping but prevents static verification and certain compiler optimizations.
- **Vectorization:** Operations apply element-wise to vectors, matrices, and arrays. For example,

  `c(1,2,3) + c(4,5,6)` $\Rightarrow$ `c(5,7,9)`

  without the need for explicit loops.
- **Lexical scoping:** R uses lexical (static) scoping with closures. Functions capture their defining environment, enabling functional programming patterns and higher-order abstractions.
- **Lazy evaluation:** Function arguments are evaluated lazily (call-by-need), allowing non-standard evaluation mechanisms that support domain-specific sublanguages.
- **Copy-on-modify semantics:** R employs implicit copying to preserve referential transparency. While this simplifies reasoning about programs, it may introduce performance overhead in memory-intensive workloads.

2.1.2 Assignment Operators

R provides multiple assignment operators with distinct scoping behavior:

- <-: Regular assignment in the current environment.
- <<-: Superassignment, which modifies the nearest existing binding in an enclosing environment.

2.1.3 Type System

R employs a dynamic type system with several foundational types, all of which are first-class objects. Unlike statically-typed languages, type information is associated with values at runtime rather than with variable declarations. Atomic Types R provides six atomic vector types:

- **Logical:** Boolean values `TRUE`, `FALSE`, and `NA` (missing).

  ```
  x <- TRUE
  typeof(x)      # returns "logical"
  ```

- **Integer:** Whole numbers, denoted with an `L` suffix.

  ```
  x <- 42L
  typeof(x)      # returns "integer"
  ```

- **Double:** Floating-point numbers (default numeric type).

  ```
  x <- 3.14
  typeof(x)      # returns "double"
  ```

- **Character:** Strings of text.

  ```
  x <- "hello"
  typeof(x)      # returns "character"
  ```

- **Complex:** Complex numbers with real and imaginary parts.

  ```
  x <- 2 + 3i
  typeof(x)      # returns "complex"
  ```

- **Raw:** Raw bytes (rarely used).

  ```
  x <- charToRaw("A")
  typeof(x)      # returns "raw"
  ```

Composite Types Beyond atomic vectors, R supports several composite data structures:

- **List:** Heterogeneous collections that can hold elements of different types.

  ```
  x <- list(42, "text", TRUE)
  typeof(x)      # returns "list"
  ```

- **Function:** Functions are first-class objects.

  ```
  f <- function(x) x + 1
  typeof(f)      # returns "closure"
  ```

  This means functions are treated as normal variables. It can be nested definition, passed as parameter and returned from a function.

- **Environment:** Hash-like structures for variable scoping.

  ```
  e <- new.env()
  typeof(e)      # returns "environment"
  ```

Type Coercion R performs implicit type coercion following a hierarchy: logical $\rightarrow$ integer $\rightarrow$ double $\rightarrow$ character. When combining types, R automatically converts to the most general type:

```
c(TRUE, 1L, 2.5, "text")  # returns character vector:
# "TRUE" "1" "2.5" "text"
```

This automatic coercion simplifies interactive use but can lead to unexpected behavior if types are not carefully managed.

2.1.4 Peculiarities

As every programming language comes with their nuances and uniqueness, R is no short of those. Below, the ones mention worthy examples

- Vectors and lists are indexed by 1. For example,

  ```
          v <- c(10,20,30) // initializes a vector of 10,20,30
          v[1]             // returns 10
          v[2]             // returns 20
  ```

- Even a scalar is represented as a vector of 1 element.

  ```
          x <- 5           // assign 5
          is.vector(x)     // return TRUE
  ```

The reason for it is to recycle the vectors easily. The language is designed for vector operations

```
x <- 5           // assign 5
v <- c(10,20,30) // initializes a vector of 10,20,30
x + v            // returns a vector of 15,25,35
```

2.2 WebAssembly

WebAssembly is a binary instruction format designed as a portable compilation target for high-level languages. In simpler terms, it's a bytecode format made to be a common-ground between web environment and different programming languages. For web environment, JavaScript is the only language that can be executed. However, with WebAssembly, commonly known as WASM, we can execute languages such as C, Rust in web environment after compiling them to WASM bytecode format. WASM main characteristics include:

- Stack-based virtual machine: WASM uses a structured stack machine with explicit control flow constructs (block, loop, if) rather than arbitrary jumps.
- Linear memory: A contiguous, resizable array of bytes for heap allocation, isolated from the host environment for security.
- Type safety: WASM enforces type safety at validation time, preventing type confusion and memory safety violations.

Chapter 3
Language Design

3.1 Design Goals

To design typed R, I tried to be as close as possible to R semantics to easily compile available R codes with wasmR. Although there are obviously inherent limitations compiling dynamic language to statically typed bytecode. Issues include mutability, typing, reflections. For example, how do we deal with

```
var <- 21  // assign integer
print(var)
var <- c(1,2,3) // assign vector
```

This is a perfectly fine code for R but compiling it to WASM brings difficulties. Therefore, like other static languages, we will add checking against mutability of different types.

- R compatibility: Preserve R's syntax and core semantics where possible
- Type safety: Static type checking with sound type system
- Ahead-of-time compilation
- A single executable WASM file, no linking required
- First-class functions: Support functional programming with closures
- Practicality: Provide essential functions for data processing (builtins)

3.2 Typed R-like Language

Syntax overview and supported language constructs.

3.3 Syntax

hello.

3.4 Type System

Primitive types, vector types, function types, and typing rules.

```
<type>          ::= <function_type>

<function_type> ::= <param_list> "->" <type>
                  | <primary_type>

<param_list>    ::= <primary_type>
                  | <primary_type> "," <param_list>

<primary_type>  ::= <builtin_type>
                  | <generic_type>
                  | "(" <function_type> ")"
                  | "function"

<builtin_type>  ::= "int" | "double" | "string" | "char"
                  | "void" | "logical" | "any"

<generic_type>  ::= "vector" "<" <type> ">"
                  | "list" "<" <type> ">"
```

3.5 Semantics

Operational semantics of expressions, control flow, and function calls.

Chapter 4

Compiler and Runtime

4.1 Compiler Architecture

Frontend, type checking, intermediate representation, and backend.

4.2 WebAssembly Code Generation

Translation of language constructs and types to WebAssembly.

4.3 Runtime System

Vector representation, memory management, and built-in operations.

4.4 Implementation Details

Key implementation choices and limitations.

Chapter 5

Evaluation

5.1 Correctness

Testing methodology and comparison with R behavior.

5.2 Performance

Execution time, code size, and runtime overhead.

5.3 Discussion

Analysis of results and trade-offs.

Chapter 6
Conclusion

Summary of contributions and future work.

Appendix A
Nějaká příloha

Sem přijde to, co nepatří do hlavní části.

Bibliography

1. HAAS, Andreas; ROSSBERG, Andreas; SCHUFF, Derek L; TITZER, Ben L; HOLMAN, Michael; GOHMAN, Dan; WAGNER, Luke; ZAKAI, Alon; BASTIEN, JF. Bringing the web up to speed with WebAssembly. In: *Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation*. 2017, pp. 185–200.

Obsah příloh

```
/
├── readme.txt ........................... stručný popis obsahu média
├── exe .................... adresář se spustitelnou formou implementace
├── src
│   ├── impl .............................. zdrojové kódy implementace
│   └── thesis .................. zdrojová forma práce ve formátu LaTeX
└── text ................................................ text práce
    └── thesis.pdf ....................... text práce ve formátu PDF
```