Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Department of Computer Science

Software Engineering Orientation

Bachelor thesis

2024

Functional language compiler to WebAssembly

Technical documentation

Noah Godel

Supervisors: Jacques Supcik
Serge Ayer

Experts: Baptiste Wicht
Valentin Bourqui

Fribourg, 27 May 2024

Version 0.1

Hes·so

Table of versions

Version	Date	Modifications
0.1	27.5.2024	First version

Contents

1. Context 1
2. Objectives 2
3. Tasks 3
4. Planning 4

Section 1
Context

The functional programming paradigm offers significant advantages for certain types of problems, such as data transformations, parallel processing, and mathematical computations. However, it has limitations, and there are many use cases where imperative or object-oriented programming paradigms are more suitable. Ideally, developers should be able to leverage the strengths of different programming paradigms within the same codebase, using the most appropriate approach for each part of the project. Unfortunately, integrating functional languages into existing codebases written in other programming languages can be challenging.

WebAssembly (Wasm) is a portable and high-performance bytecode format designed to execute code at near-native speeds. It allows code written in various programming languages like C, C++, Rust, and others to be compiled to Wasm bytecode, which can then run in environments such as web browsers and Wasm runtimes like Wasmer.

By developing a new functional programming language that compiles to Wasm, we can combine the benefits of functional programming with the performance and portability of Wasm. This approach would enable seamless integration and embedding of high-performance functional code into existing codebases written in different languages. Developers could utilize the strengths of functional programming for specific parts of their projects, while leveraging other paradigms for the remaining codebase.

To generate efficient machine code for the new functional programming language, we will leverage the LLVM framework. LLVM is a modular and extensible compiler framework that supports a wide range of programming languages and processor architectures. Notably, LLVM supports Wasm as a compilation target, allowing it to generate optimized Wasm bytecode from an intermediate representation called LLVM IR. One of LLVM's key advantages is its ability to generate highly optimized machine code for various platforms, including Wasm, ensuring optimal performance for the compiled functional language.

Section 2
Objectives

Upon the completion of the project, the following objectives will be achieved:

- **Functional Programming Language Specification**: A comprehensive specification defining the syntax, semantics, and features of a new functional programming language, incorporating key concepts such as pattern matching, first-class functions, immutable data structures, a simple type system, a module system for organizing and encapsulating code, and a minimal standard library.
- **Functioning Compiler**: A fully operational compiler capable of translating the defined functional programming language into efficient Wasm bytecode, enabling high-performance execution across various environments and platforms. While not the primary focus, the compiler should also support generating native executables, although the primary target will be Wasm.
- **Language Documentation**: Extensive documentation detailing the usage and development of the new functional programming language, including examples, and reference materials to facilitate learning and adoption by developers.
- **Integration Examples**: A collection of examples demonstrating the integration and execution of the compiled Wasm code within different programming languages and frameworks, showcasing the language's interoperability and potential for seamless embedding.
- **Embedding Demonstrations**: Practical demonstrations illustrating the embedding and utilization of the new functional language within existing codebases written in other programming languages, highlighting its ability to coexist with and complement other programming paradigms.

By achieving these objectives, the project will deliver a well-defined functional programming language optimized for Wasm execution, along with a functioning compiler, documentation, integration examples, and embedding demonstrations. While not production-ready after the 7-week timeline, the project will serve as a proof of concept and a foundation for potential further development, with the potential for future adoption and integration into codebases.

Section 3
Tasks

1. **Define Language Specification**
   1.1. Conduct research on existing functional programming languages and their features.
   1.2. Specify the syntax, semantics, and core language features, including pattern matching, first-class functions, immutable data structures, type system, and module system.
   1.3. Design the requirements and structure for a minimal standard library.
2. **Develop Compiler**
   2.1. Research the requirements and best practices for compiling to Wasm using LLVM and what language to use for the front-end.
   2.2. Implement a minimal viable compiler (lexer, parser and LLVM-IR code generation) for the new functional language with limited features.
   2.3. Configure and integrate LLVM to generate Wasm bytecode (and native code) from the LLVM-IR output.
   2.4. Optimize the compiler to generate efficient Wasm code and explore LLVM optimizations for the generated code.
   2.5. Implement the remaining language features and optimizations to complete the compiler.
   2.6. Implement a simple standard library for the language.
3. **Testing and Validation**
   3.1. Develop an automated test suite to validate the correctness of the compiler and the Wasm bytecode.
   3.2. Perform testing of code embedding and execution in different environments and platforms.
   3.3. Conduct performance benchmarking and analysis to evaluate the efficiency of the compiled code.
4. **Create Language Documentation**
   4.1. Draft comprehensive documentation covering the language syntax, semantics, features, and a reference for the standard library.
   4.2. Develop examples to facilitate learning and adoption of the language and its embedded use cases.
5. **Project Documentation**
   5.1. Write a requirements specification document outlining the context, objectives, tasks, and planning for the project.
   5.2. Prepare a detailed project report documenting the design, implementation, and evaluation of the language and compiler.

Section 4
Planning

Figure 1 shows the Gantt chart representing the project timeline and tasks.

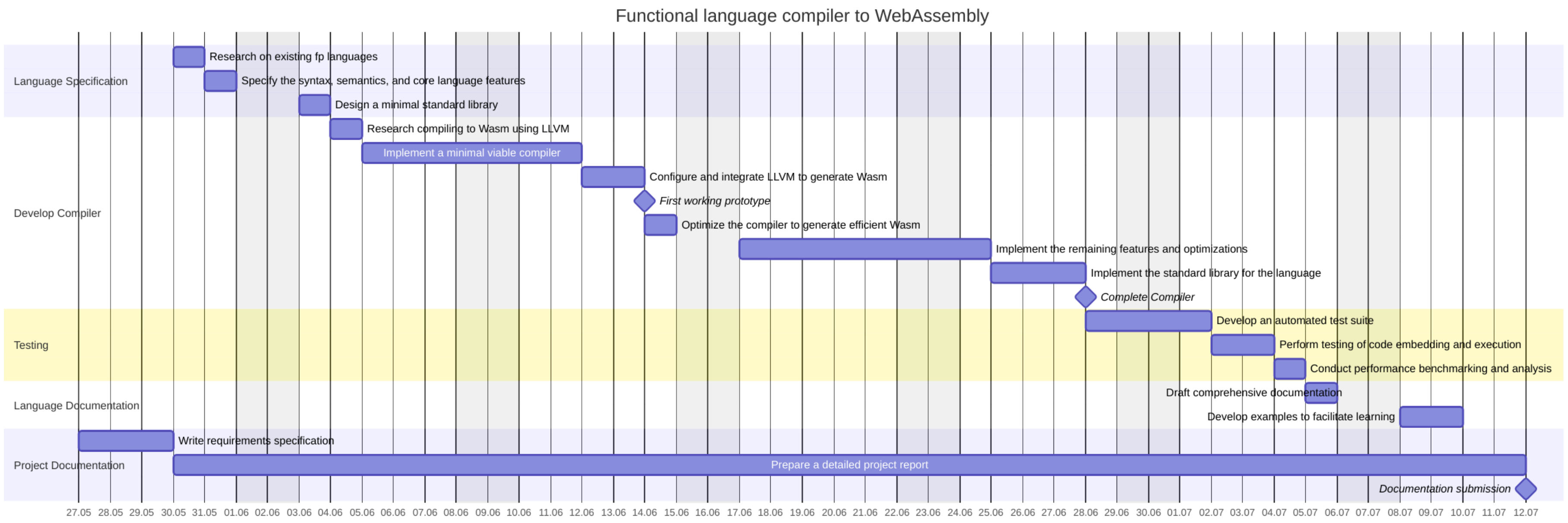

Figure 1: Gantt chart showing the project timeline and tasks.