IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 12, Special Issue 2, April 2023

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India

Haze - Programming Language

Tharun Kumar T[1], Saral Jeeva Jothi[2]

U.G. Student, Department of Computer Engineering, Velammal Engineering College, Chennai, Tamil Nadu, India [1]

Associate Professor, Department of Computer Engineering, Velammal Engineering College, Chennai, Tamil Nadu, India [2]

ABSTRACT: Haze, a new programming language written in the Go language. It is designed to be easy to learn and use while still providing strong guarantees of type safety and memory safety. The language is inspired by modern programming paradigms such as functional programming and has a concise and expressive syntax.It is designed to be run on a custom stack-based virtual machine, also written in Go. The virtual machine is designed to be lightweight and efficient, and provides a runtime environment for Haze programs that is both fast and secure. The virtual machine is built using modern compiler techniques and leverages the strengths of the Go language, including its strong concurrency support and garbage collector. The type system in Haze is based on algebraic data types, which enable powerful pattern matching and allow for precise handling of complex data structures.

KEYWORDS: Interpreter, Type Safety, Programming Language,Go Lang, Pratt Parser

I. INTRODUCTION

Haze is a new programming language that is designed to provide a simple and intuitive syntax for developers, while still offering powerful features and type safety. The language is intended to be easy to learn and read, making it accessible to programmers of all skill levels. Haze takes inspiration from popular programming languages such as Python, Ruby, and JavaScript, while also incorporating its own unique syntax and semantics. The Haze interpreter is written in Go because of its simplicity, type safety and for its performance [2].

One of the key features of Haze is its emphasis on type safety [2]. All variables in Haze are explicitly typed, which helps to prevent bugs and errors that can occur when variables are assigned incorrect values. This also makes the code more self-documenting, as it is clear from the variable names and types what their purpose and expected value ranges are.

Another important aspect of Haze is its use of a Pratt parser for syntax analysis. This allows for efficient and flexible parsing of complex expressions, as well as the ability to handle both infix and prefix operators [4]. The parser builds an abstract syntax tree that is used to evaluate the code, which allows for fast and accurate execution of Haze programs.

Overall, Haze is a promising new programming language that combines the best features of existing languages with its own unique syntax and design. As more developers become familiar with Haze, it has the potential to become a popular choice for a wide range of programming tasks.

II. DESIGN AND IMPLEMENTATION

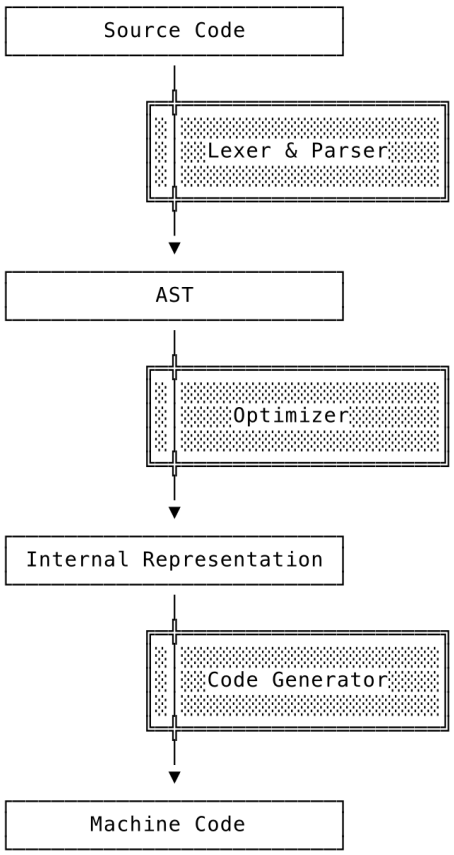
The design and implementation of the Haze interpreter consists of several parts, each responsible for performing specific tasks. These parts together parse, evaluate the Haze source code.

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India



Lexer: The lexer, also known as a tokenizer [1], is responsible for breaking the Haze source code into smaller pieces called tokens. It scans the source code character by character and groups them into logical units such as keywords, identifiers, operators, literals, etc. These tokens are then passed on to the parser for further processing.

Parser: The parser is responsible for analyzing the structure of the Haze source code, using the tokens generated by the lexer to build an abstract syntax tree (AST). It determines the precedence and associativity of operators and creates a tree-like representation of the code. The AST is then evaluated by the interpreter to produce the output [1].

Interpreter: The interpreter is responsible for evaluating the Haze source code represented by the AST generated by the parser [1]. It traverses the AST, executing the instructions represented by each node. The interpreter is also responsible for managing the scope and lifetime of variables, performing type checking and coercion, and handling control flow constructs like loops and conditionals.

REPL: The Read-Evaluate-Print-Loop is also a part of the Haze programming language specification [1]. It provides a simple user interface to interpret and evaluate Haze providing the result or errors for that matter for the source code provided.

The design and implementation of Haze interpreter is highly modular, making it easy to add new features and extend its capabilities. The interpreter is written in Go language and is highly optimized for performance, ensuring that the execution of Haze source code is fast and efficient.

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India

To implement the syntax analysis phase, the Pratt parser is used. The Pratt parser is a top-down operator precedence parser that can handle both infix and prefix operators [4]. It is simple to implement and can parse complex expressions with ease. The parser builds an abstract syntax tree (AST) that is used to evaluate the code.

In summary, the design and implementation of the Haze interpreter consists of three main parts: lexer, parser, interpreter [1]. These parts work together to provide a powerful and efficient programming environment for Haze programmers.

III. FUTURE SCOPE

The future scope of Haze includes the creation of a custom VM and a compiler to compile Haze source code, which would enable Haze to run natively on different platforms and improve its overall performance [3]. The custom VM would be designed specifically for running Haze code, providing a more efficient and optimized runtime environment compared to the current interpreter. This would result in faster execution times and a more seamless user experience. Additionally, the VM would allow Haze to be run on a variety of platforms without the need for any external dependencies [3], making it more portable and easier to use.

To enable compilation of Haze source code, an intermediate code representation would need to be generated from the AST produced by the parser. The intermediate code would be a low-level representation of the Haze code, consisting of a series of instructions that can be executed by the custom VM [3]. The compiler would then take this intermediate code and generate machine code that is optimized for the target platform, resulting in faster execution times and better overall performance.

In addition to improving the performance of Haze, the creation of a custom VM and a compiler would also open up new possibilities for the language. For example, it would enable the creation of standalone Haze applications that can be distributed and run on any platform without the need for the Haze interpreter [3]. This would make Haze a more attractive choice for developers looking to create cross-platform applications.

IV. CONCLUSION

The concept and implementation of Haze, a novel typed, interpreted language written in Go that makes use of a Pratt parser for syntax analysis [4], have been covered in this journal report. The block diagram of the Haze interpreter and the potential future direction for the development of a unique virtual machine and compiler [3] are further outlined. We are excited to investigate Haze's potential as a straightforward, expressive, and type-safe programming language as it continues to evolve.

REFERENCES

1. Ball, T. (2018). Writing An Interpreter In Go. California: IndieCS.
2. The Go Programming Language Specification. (2021, August 16). Retrieved September 27, 2021, from https://golang.org/ref/spec
3. Ball, T. (2022). Writing a Compiler in Go. California: IndieCS.
4. Pratt, V. (1973). Top Down Operator Precedence. Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 41-51.

INNO SPACE
SJIF Scientific Journal Impact Factor

SJIF: 8.423

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INTERNATIONAL CENTRE

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details