**Compiler Construction and Type Inference**

Matthew Ibrahim

Master of Science in Computer Science Capstone Project Proposal

Grand Canyon University

Professor Aiman Darwiche

January 18, 2023

**Abstract**

The evolution of software has given developers an arsenal of tools, libraries, and abstractions that allows for increased productivity and reduces the barrier of entry for new programmers to begin to develop software. Fundamentally, hardware and software has been decoupled almost entirely. Software engineers can find success without the need for a deep understanding of hardware, and software can be built on and deployed to a plethora of different platforms and architectures. To allow for such decoupling, developers must be able to write hardware-agnostic code, and various abstractions must be in place to make this possible. A specific area of improvements relates to the creation of modern programming languages, such as Go, Rust, and Python; and continuous improvements in existing ones, such as Java and C++. The increasing complexity of software and programming languages necessitates increasingly complex compilers (the software responsible for transforming source code into an intermediate, portable, or executable form) to aid developers and ensure correctness. One vital tool to aid in software development has been the evolution of type systems, in particular: the ability to provide the compiler more semantic information based on the kind of values a variable can hold. This allows a large range of bugs to be caught at compile-time, which can save time, money, and effort. In this project, the focus is on the design and construction of a compiler with a simple type system and support for type inference. Specifically, a compiler for a custom, statically-typed programming language called MattyLang will be the product of this project.

**Table of Contents**

Project Overview and Project Objectives 4

Project Scope 6

Project Completion 8

Project Controls 9

Project Cost and Schedule 10

Requirements Analysis 11

**Project Overview and Project Objectives**

**Background**

Modern programming languages have significantly improved the process of software development. Many of these languages and their ecosystems provide abundant tooling, ranging from static analysis tools, such as linters and formatters; dynamic analysis tools, such as Valgrind; to package managers, such as vcpkg. A subset of these languages are statically-typed and provide type inference (e.g., Java). This project explores the design and construction of a statically-typed language that provides type inference in order to demonstrate language design, compiler construction, and the implementation and utility of type inference in an accessible manner.

**Project Objectives**

The project encompasses the design and implementation of a compiler for a custom statically-typed programming language with support for type inference. The project will focus primarily on the process of parsing and analysis, so code generation is not a goal of this project. Execution of a source file will be done by first parsing the file into a syntax tree, ensuring the program is semantically sound, and then traversing the decorated abstract syntax tree to evaluate expressions, follow function calls, respect variable assignments, and so on. The design and implementation of the language will be simple to ensure the compiler can be used as a learning tool by others, and implemented in a language readable to most people. However, it should also be extensible and accessible to accommodate long-term goals such as the introduction of compound objects, records, tuples, and so on.

By the end of the project, a well-tested compiler for the *MattyLang 1.0.0* language will be implemented and include support block-scoped variables, functions, expressions, type checking and inference. Lexical scoping may be introduced in the future alongside closures, but are otherwise outside of the scope of this project.

**Challenges**

In order to make MattyLang useful, an interpreter or runtime must be created for the language. Given that a MattyLang runtime is not really within the scope of this project, transforming the source to Python will instead be done.

**Benefits and Opportunities**

The end result is a highly readable and extensible implementation of a compiler for a simple, procedural, statically-typed language with syntax similar to modern languages. The language will be created under an MIT license, and can be both useful as a learning tool and as a foundation for introducing a domain-specific language to a project.

Project Scope

MattyLang 1.0.0 will include support for a variety of common language constructs, such as arithmetic expressions; branching statements; block-scoped variables; and so on. Many out-of-scope features are planned to be implemented after the first major version of MattyLang, though future versions are beyond the scope of this project.

In Scope Features:

* Faultless compiler, which includes: lexical analysis (tokenization), syntax analysis (parsing), semantic analysis (symbol binding and type checking), and code generation (transforming the decorated AST to Python).
* Helpful diagnostics, including contextual information such as the line and column within the text.
* TextMate grammar for MattyLang.
* Unit/component tests to meet high code coverage.
* Primitive types and literals: Nil, Bool, Real, String.
* Arithmetic, logical, and relational expressions.
* Conditional and branching statements.
* Function definition statements and function call expressions.

Out of Scope Features (many of these will likely be implemented after the first major version of MattyLang):

* Complex features such as closures, compound types, and generics are a non-goal.
* No MattyLang runtime will be developed at this time, as MattyLang compiles to Python.
* No language server, linter, or formatters will be developed at this time.
* MattyLang module/library support will not be developed at this time.

**Work Breakdown Structure**

1. Project setup (Complete): initialize repository, tooling, and CI

2. MattyLang v0.1 (In Progress): expressions (arithmetic, relational, logical), primitives (Nil, Bool, Real, String), and types.  
3. MattyLang v0.2: branching statements: if, while, break, continue. Depends on v0.1.

4. MattyLang v0.3: functions, function calls, return statement, function types. Depends on v0.2.  
5. MattyLang v1.0: additional documentation, tests, fixes, improvements. Depends on v0.3.

Project Completion

The measures that will be used to calculate project success are:

1. Completion: how much of MattyLang v1.0 has been developed?

2. Correctness: how closely does the implementation adhere to the specification?

3. Documentation: how well documented is the codebase, and how well documented is the specification of MattyLang?

4. Output: how meaningful are the diagnostics produced if there are errors?

Project Controls

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RISK MANAGEMENT | | | | |
| **Event Risk** | **Risk Probability**  **(high, medium, low)** | **Risk Impact** | **Risk Mitigation** | **Contingency Plan** |
| Cannot complete v1.0 in time. | low | Project is in an incomplete state, though v0.3 is acceptable. | Ensure proper time management. | Ensure features in v0.1, v0.2, and v0.3 are developed as much as possible before extra code/documentation improvements. |
| Presentation issues | medium | Less acceptable presentation | Ensure no technical problems ahead of time, microphone quality should also have been tested, and some notes should be used to guide the presentation. | Keep recording the presentation over and over until happy with result. |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ISSUES LOG | | | | | | | | |
| **ID** | **Description** | **Project Impact** | **Action Plan/Resolution** | **Owner** | **Importance** | **Date Entered** | **Date to Review** | **Date Resolved** |
| 1 | Runtime is out of scope but required. | Significantly increases implementation time | Instead of implementing an interpreter/runtime, transform AST to Python. | Matthew | *VERY* | *1/17/2023* | *1/18/2023* | *1/18/2023* |

Project Cost and Schedule

No cost calculation needed.

**Schedule**

1. Project setup (Complete): initialize repository, tooling, and CI

2. MattyLang v0.1 – Complete by January 26, 2023  
3. MattyLang v0.2 – Complete by February 2, 2023

4. MattyLang v0.3 – Complete by February 9, 2023  
5. MattyLang v1.0 – Complete by February 16, 2023

Requirements Analysis

**Feature Version Description**

program 0.1 root node

block 0.1 sequence of statements

statement 0.1 to 0.3 unit of execution

- function\_definition 0.3

- variable\_definition 0.1

- return\_statement 0.3

- if\_statement 0.2

- while\_statement 0.2

- break\_statement 0.2

- continue\_statement 0.2

- call\_expression 0.3

- assignment\_expression 0.1

expression 0.1 to 0.3 statement operand

- '(' expression ')' 0.1

- nil\_literal 0.1

- bool\_literal 0.1

- real\_literal 0.1

- string\_literal 0.1

- identifier 0.1

- call\_expression 0.3

- assignment\_expression 0.1

- unary\_expression 0.1

- binary\_expression 0.1

type 0.1 to 0.3 value/expression kind

- Nil 0.1 unit type, only **nil**

- Bool 0.1 either **true** or **false**

- Real 0.1 IEEE 754 double

- String 0.1

- Function Type 0.3