Final Project
Computer Science, Bar Ilan University

Noam Kaplinski and Ari Feiglin

1 Introduction

Our idea is to create a programming language utilizing a (novel?) algorithm we have developed. The algorithm allows for parsing and lexing to be done at runtime on an on-demand basis: tokens are only read and expressions evaluated when they are needed, as opposed to an entire line being read, lexed, then parsed into an AST (abstract syntax tree).

Instead, the program is parsed in a linear-esque fashion, similar to macro languages like TEX. Similar to TEX, the language will expose internal functionalities to the programmer, allowing them to alter the behavior of the lexer and reducer (parser).

The language will be a dynamically typed multi-paradigm language, with syntax similar to that of JavaScript. Meaning it will support both procedural and object-oriented programming. The language will also allow for metaprogramming: the programming of the language itself (its lexer and parser), by directives which alter how the language interprets symbols, expansion, etc.

2 The Structure of the Project

The project will have two main components: a theoretical part, and a practical part. The theoretical part will be a paper describing the algorithm and the implementation of the practical part, which will be an interpreter for a language we will create.

TODOs for the theoretical part:

(**1**) Finalize how the algorithm works. This means potentially altering the definition of a β-reducer to allow for OOP. Another issue is that of scoped variables (alter the definition of a state) and closure.

(**2**) Come up with a specification for the language.

(**3**) Come up with an initial β-reducer and priority function for the specification.

(**4**) Write a coherent paper explaining the goals of the language and algorithm, as well as how it was implemented.

TODOs for the practical part:

(**1**) First, I think it may be a good idea to write a prototype in an easy language like Python.

(**2**) Implement the initial β-reducer as well as the extension rules for β-reducers.

(**3**) Implement a lexer (all it needs to do is read the next token and convert it to a token and priority; I have a decent algorithm for this already).

(**4**) Implement a final interpreter in C++.

These two components are inextricable, so we will have to work on both simultaneously probably. For example, while writing the specification we should understand how the initial β-reducer should function on it, and perhaps implement it in Python to verify that it works.

3 Steps

Some steps we maybe should follow:

(**1**) Get together, explain the algorithm in depth. Explain what problems I'm working on currently, potential solutions.

(**2**) Work on finishing up the algorithm, do so by extending the specification and initial β-reducer for the program. This will probably be the hardest step.

(**3**) Document the algorithm, specification, and initial β-reducer in the paper.

(**4**) Implement a Python version of an interpreter.

(**5**) Document the implementation in the paper.

(**6**) Translate from Python to C++.

(**7**) Write a standard library for the language?