Design Document: Untyped Lambda Calculus Interpreter in Python

Ryan Puglia

May 7, 2025

Overview and Goals

The goal of this project is to implement a simple interpreter for untyped lambda calculus in Python. The programme should support the following:

- a function for computing the set of free variables in a lambda expression
- a function for substituting , in a lambda expression, all occurences of the formal parameter with another lambda expression
- a function for performing α-conversion
- a function for performing β-reduction
- a function for reducing a lambda expression to its normal form
- a function for pretty-printing a lambda expression

Language Choice

There were multiple choices which were given to implement a lambda calculus interpreter, namely Python, Java, Javascript and Typescript. Due to having more experience with the Python programming language, I decided to use it for this project.

Design/Implementation

The programme was implemented using "Test driven development". Implementations of the programme's functionalities were done in the following order:

1. Write the function signature (skeleton)
2. Write (failing) unit test for expected results and behaviour
3. Implement the functionality until the unit test passes
4. Refactor if needed

Object-oriented programming principles in Python were chosen to be used for this programme. This approach allows for clarity and extensibility. The programme makes use of method overrides, such as equality checks and pretty printing.

Core Components

AST Classes The interpreter defines three classes inheriting from a base class `LambdaExpression`:

- `Var(name)`: A variable.
- `Abs(param, body)`: A lambda abstraction.
- `App(func, arg)`: An application.

Each class implements equality and pretty-printing methods.

Parser and Lexer The `lexer` function tokenizes input strings into symbols like `(`, `)`, λ, `.`, and identifiers. The `Parser` class uses recursive descent to produce ASTs from token sequences. Only the Unicode λ character is accepted for abstractions. The parser class includes several helper methods. These helpers methods consist of `peel` to peek at the next token without consuming it, `consume` to consume the next token and `parse_atom` to parse a single variable or expressions withing parentheses. These helper methods are used in the parse_expr method.

Substitution and Alpha Conversion The `substitute` function performs capture-avoiding substitution. `alpha_conversion` allows renaming of bound variables.

Evaluation and Normalisation `beta_reduction` applies one step of normal-order reduction. `normalise` applies beta reductions until a normal form is reached (if one exists).

Evaluation Strategy

The interpreter uses **normal-order reduction**, which always reduces the leftmost, outermost term first. This strategy guarantees finding a normal form if one exists.

Error Handling

- Invalid syntax in input expressions raises `SyntaxError`.
- File errors (e.g., file not found) are caught and displayed.
- Unexpected errors are caught and reported without crashing.

Testing

Unit tests verify each major component:

- Free variable computation
- Substitution and alpha conversion
- Beta reduction and normalisation
- Lexer and parser correctness
- Pretty-print consistency

Tests are written using Python's `unittest` module.

Github repository

The github repository which contains all of the source code and some example .lam files can be found using the following link:
https://github.com/puglia-ryan/Lambda-Calculus-Interpreter-in-Python