CS 51 Project Write Up

Arseniy Zvyagintsev

April 2022

1 Introduction

In this paper I am going to demonstrate two extensions for my CS51 final project - lexical environment semantics and atomic data types

2 Lexical Environment Semantics

In this extension I have implemented and tested the lexical environmental semantics (see eval_l function in the evaluation.ml file) as suggested in the section 21.4.2 in the textbook. To switch to the lexical semantics, a user should indicate "let evaluate = eval_l ;;" in the end of the evaluation.ml file.
The main feature of this semantics is environments' closures used when a function is defined. When the function is defined, the program creates a duplicate of the current environment (it is important to make a structural but not physical duplicate as otherwise the closed environment might be externally changed) and stores it with the function itself. When the function is applied, the program runs the function not in the current environment but in the closed (stored) environment. This lexical environment semantics evaluation is equivalent to the substitution semantics one (except for function evaluation which in the lexical semantics evaluates to a closure) though it uses deeper ideas of environmental closures discussed above.
The main challenge of implementing the lexical environment was recursive let. In order to do that, I assign the Unassigned value to the name of the let expression and add it to the environment as discussed in the 21.4.2 section in the textbook.
To assure yourself that these semantics are equivalent, take a look at the file test.ml where I tested expressions in all three implemented semantics. We can see that dynamics semantics acts differently in some cases than substitution or lexical ones.

3 Atomic Data Types

In this extension I have implemented two most important atomic data types - float and string. The implementation required several steps.

1. Adding data types and their operations in "type binop" and "type expr" in the files expr.ml and expr.mli

2. Extending all functions to support these data types in the files expr.ml and evaluation.ml. Fortunately, I wrote a lot of helping functions, so these edits took only a couple of lines, primary in the helping functions binop_match (returns the result of the binary operation), exp_to_concrete_string and exp_to_abstract_string

3. Detecting new elements of the input stream using miniml_lex.mll functionality. For this, I extended the sym_table and added symbolic signatures for strings (the string should start and end with ") and floats (the string should consist of digits with a dot in the middle)

4. Parsing correctly new elements in the miniml_parse.mly file. For this, I added and handled new tokens corresponding to string and float data types

Usage of this extension is very intuitive. The user should just write the strings within the " symbols and floats with a dot symbol. Symbolic operations in my extension are similiar to OCaml's ones: $\wedge$ is string concatenation, +., -., *., $\sim -$. are numerical operations on the float type. Comparison operators (=, ¡, and added ¿) are extended as well to support float and string types.