CSC 254 Assignment 3: Interpretation

Prikshet Sharma

October 18, 2019

1 To run

In the Ocaml interpreter run

```
# load "str.cma"
# use "interpreter.ml"
```

2 Overview

We have implemented the syntax tree generation and the interpretation phases of the calculator language in this assignment. Some extra functions were created like add, sub, great, less etc. Furthermore, get_op was implemented, which is used in interpret_expr to get the right operator according to the operator in the syntax tree. For extra credit we have implemented unused-variable warnings and C code generation.

3 Extra Credit 1: Unused Variable Warnings

. Whenever a variable is initialized or read, it is put into the memory as a tuple. The tuple holds three values: a boolean, which indicates whether it has been used or not, a string, represents is the id of the variable, and a value type, which holds the value of the variable. Whenever the variable is used in an expression, we set its corresponding boolean value to true. Of course, since OCaml lists are immutable, we cannot really set the value. Instead we return a new list, that contains only the requisite id's tuple altered.

After we return the memory using the interpret function applied to a calculator language program, we check each element of this memory for tuples that have a boolean value of false. For each of these tuples, we print a warning message that displays the id as well.

This is implemented in the get_used function shown as follows:

```
get_unused (m:memory): string =
match m with
| (b, i, v) :: rest -> (if b then "" else "\nWarning: " ^ i ^ "
   not used") ^ (get_unused rest)
| [] -> "\n"
```

This function is called from a modified interpret function:

```
interpret (ast:ast_sl) (full_input:string) : string =
let inp = split (regexp "[ \t\n\r]+") full_input in
let (_, m, _, outp) = interpret_sl ast [] inp [] in
(fold_left (str_cat " ") "" outp) ^ "\n" ^ (get_mem m) ^
   (get_unused m)
```

4 Extra Credit 2: C Code Generation

Most of the syntax between both these languages has a simple one-to-one mapping. For example "a := 4" becomes "a = 4;" etc. The only thing that we must do is to declare all the variables. For each scope, we declare all the variables that are used in that scope that are not already used in the surrounding scope. The code is also properly indented.

4.1 Limitation

We scan the whole list of ids when declaring ids in the beginning. This has a complexity of O(n), where n is the number of ids. This can be reduced to O(1) with a hash table, which wasn't implemented.