COL226 Assignment 4

Avval Amil
Entry Number : 2020CS10330

27th March 2022

1 Introduction

In this assignment I have made a VMC machine for evaluating programs in the WHILE programming language. The overview is fairly similar to the one specified in the assignment pdf, but I have included some design decisions where I felt they were necessary or would speed up the evaluation of the program. Any program written in the WHILE language should be compiled and evaluated by this VMC machine, with input and output facilities provided. The code is explained in greater detail in the comments of the file "test.sml".

2 Differences with respect to Assignment 3

This assignment required the implementation of the previous assignment, and I have made some changes in my implementation of the scanner and parser. The grammar has been modified to reduce and remove shift-reduce and reduce-reduce conflicts, and type-checking has also been implemented in the "wast.yacc" file itself. The updated files are in this submission, so they are slightly different to the ones in the submission of assignment 3. Naturally, the datatypes are also slightly modified to accommodate for the changes in the grammar. The "wast.lex" file, however is the same. In the hash table, the key value pairs are as follows - the variable names (strings) are the keys, and the values can be either 0 (standing for int type) or 1 (standing for bool type). Using this, a 'type' is assigned to each variable and expression and thus type errors (as well as 'use-before-declaration' errors as lookup is needed from the hash table) prevent any incorrect programs from being compiled.

3 Files and Overview

The submission consists of two kinds of files - files which are written by me, and files which the SMLNJ Compiler Manager creates when it is used (e.g. "wast.yacc.desc"). Further, in the files that are written by me, there are also different types of files. The file "datatypes.sml" defines the user-defined datatypes.

The file "glue.sml" contains the gluing code to connect all the different files and allow them to communicate. The file "test.sml" contains all the code of the VMC machine that is required in this assignment. Everything from the FunStack structure to the Vmc structure are defined in this, along with the transition rules for the configurations. The "wast.cm" file is used to invoke the SML Compiler manager to compile the files. Finally, the "wast.lex" and "wast.yacc" files include the code for the lexer and parser of the WHILE programming language respectively.
I have also included three basic sample programs written in the WHILE programming languages to calculate the absolute value of a number, factorial of an integer, and sum of an AP which can be tested to check that they do produce the required final configurations expected. This was done just for the purpose of testing my code, but I have included them in the submission nevertheless.

4 Code and Design Choices

The FunStack structure has been implemented using the inbuilt SML lists as their behaviour resembles that of the abstract stack datatype. The "test.sml" file contains the main code of this assigment. First the FunStack structure is defined, and then a few helper functions are defined to ease the conversion of the AST to postfix form. All functions are accompanied with comments defining what they do.
Here, an important design choice was made, in the postfix form of an AST. For every 'command' constructor (SET, WRITE, READ, ITE and WH), I have included a special start symbol "$" in the postfix form of the AST, so that a kind of "lookahead" can be implemented using parenthesis matching (with "$" as the starting symbol and any one of the command constructors as the end symbol). This is necessary to deal with cases of While loops and ITE statements as we need to know that an ITE statement is coming in advance without executing whatever command comes blindly. Similarly for While loops as well. Note that empty commands have to be handled differently, as they don't end at a command constructor and hence I've made separate cases for empty commands.
Another smaller design choice made is that in the string form of the constructors, I've appended the string "@" at the front. This is done because many constructors have names which are not keywords, and hence a variable may be named the same as that constructor, causing confusion as to whether a variable or a command constructor is present in a position. Appending "@" to constructors solves the problem as no variable can start with an "@".
After converting to postfix form, the transition rules for configurations are also encoded. A hash table is used to hold the indices of variables along with their names, and a function "createVMC(filename)" creates a VMC machine initialized according to the WHILE program in the file 'filename'. A function almost identical to the one for transition rules is also implemented to find the value an expression evaluates to. This could have been done on any stack, including the original V stack, but I just did it on an empty stack for sake of brevity of

code. At last, the Vmc structure is defined along with the functions needed. The ”toString” takes a VMC configuration as the input and gives a 3-tuple of string lists representing V, M and C respectively. The ”postfix” function takes an AST as the input and gives out a list of strings which represents the postfix form. The ”rules” function takes in a configuration of a VMC machine and outputs another updated configuration. The ”execute” function takes in a string representing the file name where the WHILE code is present, and gives out the final configuration of the VMC machine corresponding to that of the code in the source file.The types of inputs and outputs of these functions are all specified in the signature itself.
Comments can be also be referred for information.

5 Compilation Instructions

Let's say that a program of the WHILE language that needs to be compiled is present in a file ”program.while” in the same directory as the rest of the submission. Then to compile the contents of that file and look at the final vmc, the following steps should be followed -
1) Open the directory which contains all the submission files in the terminal, and make sure the ”program.while” is present in that directory and open the SML interpreter.
2) Type CM.make ”wast.cm”; to compile the compiler.
3) Type Open DataTypes; to make the user defined datatypes accessible.
4) Type use ”test.sml”; to use the file in which everything is defined for this assignment.
5) Type Vmc.execute(”program.while”); to get the required final configuration (if one exists) of the program written in ”program.while”.

6 References and Acknowledgment

All of the code that is written is entirely mine and only syntax and other documentation has been referred to on the internet. The references used are as follows -

1) https://smlfamily.github.io/Basis/
2) http://rogerprice.org/ug/ug.pdf
3) https://www.smlnj.org/doc/smlnj-lib/Util/str-HashTable.html