KRISHNA KANT GARG (kgarg8@uic.edu)
SUTANU KUMAR GHOSH (sghosh34@uic.edu)

CS-476 PROJECT REPORT (FALL 2018)
DSL FOR MATRIX IMPLEMENTATION USING OCAML

We implemented basic Ocaml types, syntax and function to perform almost all the basic operations that can be done on a matrix. The basic matrix operations include Assignment of a matrix, Addition, Subtraction, Product, Scalar multiplication of 2 matrices, Transpose of a matrix, Determining the size of a matrix etc. We also implemented to execute a sequence of operations involving matrices. Our implementation can be run either by running the *step_cmd* function or the *big_step* function, both of which gives us an output in of type *(stmts * state) option*. The *stmts* type in our program has several constructors each of which performs an operation on matrices or a single matrix based on the name of the constructor. The program can also compute the size of the matrix.

The *addMatrices* function is a simple function which takes 2 matrices as input which are basically lists of intlist and produces the result as the addition of the 2 matrices. Likewise, the *subtractMatrcies*, and *multMatrices* are also similar functions which produce the result as subtraction and multiplication of 2 matrices respectively. The next function *scalarMult* takes an integer and a matrix as input and produces the scalar multiplication. There are also a few functions which perform some varied multiplications on 2 matrices. The next function and perhaps the most challenging one is the *transpose* function. Because, when we do a *transpose* of a matrix the dimensions of the matrix get flipped. The *transpose* function produces the exact result of transposing a matrix, which later can be implemented for further matrix operations. These are in brief the functions which we used in our program.

We implemented the Ocaml *Printf* module to print all the resulting matrices in a proper format. For printing the results in the output, we used the *Put* constructor of the type *stmts*. To extend these functions, we also implemented the *stmts* type to have a different section of constructors which generally will deal with matrices with their names. Such as, the *Assign* constructor will do the following: *[A]=[B]* apart from the normal operation like *[A] = [4;5;6],* which is done by *Assign2* constructor. Similarly, the *Add, Sub, Mul, Prod, ScalarMult, Transpose and Size*- all these constructors can deal with the above-mentioned matrix operations which deal with matrices declared and used by their name only. So, when implementing *Add* or *Sub* we can do the following: *[A] = [B] + [C] or [A] = [B] - [C]* given that some values have already been assigned to the matrices *[B]* and *[C].* The other constructors do a similar thing of performing operations by using the matrix names only. We also implemented the *update* function which keeps track of all the matrix identifiers to its respective value. To conclude, we are hoping to take ahead this project to build a Theorem Prover for proving some of the interesting properties on matrices using Coq programming language.

How to run the files:

1 #use “driver.ml”;; (*Contains actual source code*)
2 #use “testcase1.ml”;; (*Contains test cases for testing every stmt individually*)
3 #use “testcase2.ml”;; (*Contains single test case of 29 lines*)