COSC 5010. Formalizing the JVM in ACL2. Spring 2004
Examination
Due Thursday 6 May 2004

- Answer these questions individually. You may only use the book, the homework, the class notes, the class website, and the ACL2 documentation. You may ask for help only from the instructor.
- Email your solutions to me at `cowles@uwyo.edu`.
- The subject line should identify the problem set, i.e. Exam Question 4.
- The body of the message should be a Lisp readable file in simple ascii text format.
- The first few lines of the body should contain your name.

Exam Question 4. Define, in ACL2, a compiler that given an expression generates code to evaluate the expression for the machine M0. Use ACL2 to prove that your compiler is correct.

Expressions. The expressions are built out of symbols, integers, the binary functions `+`, `-`, `*`, and the unary functions `inc` (increment), `dec` (decrement), `-` (unary minus), `sq` (square), and `dbl` (double). The expressions are in LISP prefix notation. The values of the symbols are given in an alist that relates symbols to integers.

- Write an ACL2 Boolean-valued recognizer for expressions. `(Exprp x)` returns `t` or `nil` depending on whether or not `x` is one of our expressions.
- Write an ACL2 function that evaluates an expression. `(Eval exp alist)` returns the value of the expression `exp` using `alist` to determine the value of any symbol in `exp`.

`Compiler.` The M0 program generated by `(compile exp)` computes and then places the value, of the expression `exp`, on the top of whatever M0 stack was present before execution of the compiled program.

Step Counter. The M0 interpreter `run` has two input parameters, a state and a number of steps. Write an ACL2 function that counts the number of steps required to drive a compiled program to completion. `(Nbr-of-steps exp)` returns the number of steps required by the M0 program `(compile exp)` to evaluate the expression `exp`.

Correctness. State and prove in ACL2 that your compiler is correct.

Challenge. `This part is not required!` None of the M0 code generated above by your compiler requires the use of any branching or goto instructions. Add the unary function `abs` to our expressions, requiring the modification of `eval`, `compile`, the correctness proof, etc. `(Abs int)` returns the absolute value of the integer `int`.

Hint: Read Section 10.6, pages 166–171, of the ACL2 book.
The `loader` mentioned in class is not needed. Its functionality is essentially that of `modify` plus `nbr-of-steps`.