

Program Verification

Exercise Sheet 8: The Boogie Intermediate Verification Language

Assignment 1 (Modularity)

Give an example of a Boogie program, including a (non-recursive) procedure p such that:

1. The program includes at least one call to procedure p .
2. Each call to p in the program is made in a state in which the precondition of p is provably true.
3. If each call to p were inlined, the states at the end of each inlined copy are each guaranteed to satisfy the postcondition of p .
4. Procedure-modular verification of p will not succeed.

Write a variant of your program which includes an `assert` statement immediately after a call to p , such that the `assert` statement will result in a verification error for procedure-modular verification, but not if the definition of p is inlined for the call.

Assignment 2 (Verification)

Consider the `n_reverse.m` procedure declaration from slide 185. Provide an implementation of this procedure. You should check that it verifies (e.g. you could test your implementation via the web interface at <http://rise4fun.com/boogie>).

Assignment 3 (Well-Definedness Conditions)

Consider the operation $\text{def}(e)$ discussed (and partially defined) on slide 187. Suppose that we are modelling integer arrays using the definitions on slide 189, and want to *enforce that array-index expressions index the array with its bounds*, as a well-definedness condition.

Write down appropriate cases of the $\text{def}(e)$ construct for:

1. an array-index expression $e[e']$
2. an equality expression $e_1 == e_2$
3. a negation expression $!e$
4. a disjunction expression $e_1 || e_2$