

COMP2411 Programming Assignment — Requirements
Due Date: Friday June 9

Model checking is a technique for program verification that involves conducting a search through the space of possible executions of a program looking for a counter-example to the specification of the program. In general, it is not possible to explore all the possibilities in a finite amount of time, but for programs and specifications expressed in restricted languages model checking is a feasible verification technique. It has proved particularly useful for the analysis of computer hardware designs and for the communications protocols used in computer networks. (See the chapters on this topic in the textbook for a discussion of the type of model checking that has proved most successful.)

This assignment concerns model checking for a small (toy) programming language designed for manipulating boolean variables, and a simple language for expressing specifications, viz., propositional logic. The goal is to develop a logic program (called “the system” below, to distinguish it from the programs that are its input) that reads in a program in this language, together with its specification in the form of an initial condition and a final condition expressed in propositional logic. The task performed by the system is to

1. check that the program and its specification are grammatically well formed: if not, report an error;
2. search for a counterexample showing that the program does not satisfy the specification. If one can be found, the system should return the counterexample. If an exhaustive search has been conducted and there is no counterexample found, then the program satisfies its specification.

A counterexample is an assignment of boolean values to the variables satisfying the initial condition, such that after executing the program from these initial values, the final condition does not hold.

The main predicate of your system should be the predicate

`check(Filename, Input, Output)`

where `Filename` is a string giving the name of the file containing the program and its specification, and `Input` and `Output` are assignments of truth values to the variables manipulated by the program. The fact `check(Filename, Input, Output)` should be true if

1. The initial condition in `Filename` is *true* for the assignment `Input`.
2. The program in `Filename`, when run with the initial values of the variables given by `Input`, halts with the variables with values `Output` at the end of the computation.
3. The final condition in `Filename` is *false* for the assignment `Output`.

Your system should work for the following modes:

1. `Filename` is given and `Input` and `Output` are variables.
2. `Filename` and `Input` are given and `Output` is a variable.
3. `Filename`, `Input` and `Output` are all given.

An assignment of truth values to the variables is to be represented by a list of pairs of the form (`variable`, `truth-value`). The following is an example of a program and its specification:

```
variables a,b,c,d ;

initialcondition ((not(a) /\ b) /\ (c /\ d))

finalcondition not(b) ;

begin

  a := b ;

  c := (d \/ b) ;

  if c then begin

    b := (a -> c) ;

    c := b

  end

  else c := a

end
```

A program begins with a statement declaring the variables that may be used in the program or the specification statement. All variables in this language have type `boolean`. The declaration is followed by an initial condition, and a final condition, expressed as formulas of propositional logic. A formula is constructed using the operators `\/` (`or`), `/\` (`and`), `not` (`negation`) and `->` (`implies`). This is followed by the program statements, between `begin` and `end`. The most basic type of program statement is an assignment statement, of the form

$$\langle \text{variable} \rangle := \langle \text{formula} \rangle.$$

This means evaluate the formula and assign the resulting truth value to the variable. The semicolon “;” is used to separate program statements. The only

other programming constructs in the language are the `if then else` statement and the `begin end` block. The condition in the `if` part can be any formula of propositional logic. Since there are no loops, programs are guaranteed to terminate.

The example program does not satisfy its specification. (Note that the only assignment satisfying the initial condition is `[(a,false), (b,true), (c,true), (d,true)]` and when the program is started from this initial state it terminates in the final state `[(a,true), (b,true), (c,true), (d,true)]`. Thus these assignments should be returned as the values of `X` and `Y` respectively when we call `check(Filename,X,Y)` and `Filename` contains this program.

A perfect solution to the problem as stated above will earn a maximum 80%. For the additional 20%, you must extend the program to handle the case of while loops. A while statement has the form `while <formula> do P` where `P` is a single statement or a `begin end` block. Such a statement repeatedly performs this statement or block while the formula remains true. Note that now it is possible for the program to get into an infinite loop and fail to halt. In addition to the type of counter-example to the correctness of program with respect to the specification discussed above, we now have another type of counter-example:

- **Input** is an assignment satisfying the initial condition such that when the program is run from this initial state, it gets stuck in an infinite loop.

Thus, your system will need to check somehow for non-termination to do handle this extension.