MI/MEI – MFES – Formal Software Verification

Assessment Test – June 20, 2011

Note to students: you are allowed to answer the questions below in either Portuguese or English.

1. Consider the block of code:

```
if (x<0) then x := -x else skip;
if (c>0) then c := c-1 else skip
```

   (a) Generate its verification condition with respect to the precondition true and the postcondition $x \geq 0$.

   (b) What else should be added to the specification to fully describe what the code does?

2. A problem of the standard method for generating verification conditions has to do with efficiency. Consider a program consisting of *a sequence of conditional commands*, and for the sake of simplicity assume that each conditional contains only atomic commands. Show that the verification condition of such a program with respect to a specification (P, Q) contains a number of copies of the postcondition Q that is exponential on the size of the program.

3. The efficiency problem of the previous question is bad not exactly because of the VC generation, but because of the automated proofs. For instance for the simple VC of exercise 1 an SMT solver would essentially have to prove 4 different formulas, all of them involving one copy of $x \geq 0$. For programs without iteration, the solution to this problem is to first convert the program to an equivalent *passive form*, in which at most one assignment is made to each variable in each execution path. The trick is to use several indexed versions of each variable. We then define the formula $\mathsf{F}(S)$ of a command block S as follows:

$$\begin{array}{rcl} \mathsf{F}(\textbf{skip}) & = & \mathsf{true} \\ \mathsf{F}(x := e) & = & x = e \\ \mathsf{F}(\textbf{if } b \textbf{ then } S_t \textbf{ else } S_f) & = & (b \wedge \mathsf{F}(S_t)) \vee (\neg b \wedge \mathsf{F}(S_f)) \\ \mathsf{F}(C\,;\,S) & = & \mathsf{F}(C) \wedge \mathsf{F}(S) \end{array}$$

   And then weakest preconditions can be calculated simply as follows

$$\mathsf{wp}(S, Q) \equiv \mathsf{F}(S) \rightarrow Q$$

   A passive version of the previous example would be

```
if (x0<0) then {x1 := -x0; x2 := x1} else x2 := x0;
if (c0>0) then {c1 := c0-1; c2 := c1} else c2 := c0
```

   And its specification would now be $(\mathsf{true}, \mathsf{x2} \geq 0)$.

   Generate the verification condition for this passive program using this method. Your VC should contain a single copy of the postcondition.

   (Don't worry if it is not obvious why the above method indeed generates the weakest precondition of a passive program, you are just asked to apply the method.)

4. Write a simple ACSL-annotated C program that you think illustrates conveniently the features of a tool like Jessie for the verification of safety and functional properties (try to avoid examples very similar to those seen in class). Explain the role of each annotation.

5. Recall the student talks given in the last three lectures of the course. Select one of these talks (obviously not yours!) and produce a summary of the area covered in that talk, and an assessment of its relevance for software verification (about 1-2 pages).