Study Exercises

The following are on pages 299–301 of the text.

Exercise 1.
Huth & Ryan’s Exercise 4.2.2 (p. 299) describes a for-statement like that found in C, C++ and related languages.

(a) Write a program equivalent to
\[
\text{for ( } C_1; B; C_2 \text{ ) } \{ C_3 \},
\]
which uses only while and if-then-else as control statements.

(b) Based on your equivalent code (or by direct arguments), give a deduction rule appropriate for the partial correctness of for-statements, of the following form.

\[
\frac{\text{? ? ? ?}}{\{ P \} \text{ for } (C_1; B; C_2) C_3 \{ Q \}} \text{(partial-for).}
\]

You will need to refer to a loop invariant.

Exercise 2.
Any or all of Exercises 4.4.13–20, pp. 301–302, in Huth & Ryan.

Exercise 3.
An advanced question, for those wishing to go further.

Show that the triple
\[
\langle \text{true} \rangle \\
A[A[2]] = 3 \\
\langle A[A[2]] = 3 \rangle
\]
does NOT hold under partial correctness.

What is the proper pre-condition for the above code, to satisfy the post-condition? Does a nested array assignment (like the above) obey the array-assignment rule, or is an augmented rule required?

Exercise 4.
Show that the decision problem of whether an arbitrary propositional formula is satisfiable is decidable.

Exercise 5.
Show that the decision problem of whether an arbitrary propositional formula is a tautology is decidable.

Exercise 6.
Show that the following problem is undecidable:
Given a Hoare triple \(\downarrow \alpha \downarrow C \downarrow \beta \downarrow\), is it satisfied under partial correctness?

That is, show that an algorithm to decide partial correctness would provide an algorithm to decide the Halting Problem.

Hint: Explain how, given a pair \((P, I)\), you can specify \(\alpha, C\) and \(\beta\), such that the partial correctness of \(\downarrow \alpha \downarrow C \downarrow \beta \downarrow\) relates to the halting (or not) of \((P, I)\).

**Exercise 7.**

*This goes a bit beyond what we covered in class, but you should be able to work through it.*

Imagine having two Scheme programs with the specifications below. Each of them takes a single argument, a positive integer, and always halts, with output a pair consisting of a program and an input to that program.

i) A call to \((\text{enumHalting } m)\), where \(m\) is a positive integer, produces a pair \((Q, J)\) such that \(Q\) halts on input \(J\).

For every pair \((Q, J)\) such that \(Q\) halts on input \(J\), there is some positive integer \(n\) (a different integer for each pair) such that \((\text{enumHalting } n)\) returns output \((Q, J)\).

ii) A call to \((\text{enumLooping } m)\), where \(m\) is a positive integer, produces a pair \((Q', J')\) such that \(Q'\) runs forever on input \(J'\).

For every pair \((Q', J')\) such that \(Q'\) runs forever on input \(J'\), there is some positive integer \(n'\) (a different integer for each pair) such that \((\text{enumHalting } n')\) returns output \((Q', J')\).

Note that every pair \((Q, J)\) is a possible output of one of the two programs, but never of both.

(a) Design an algorithm by which a Scheme program, with access to the above programs, could solve the Halting Problem; that is, could answer correctly any question of the form

Does program \(P\) halt on input \(I\)?

(b) Explain why your algorithm demonstrates that the programs \texttt{enumHalting} and \texttt{enumLooping} cannot both exist.

(c) In fact, one of the two programs does exist. Can you guess which one? Can you describe an algorithm for it? (Giving an algorithm is definitely an advanced question. Don’t worry if you can’t solve it completely.)