Prof. Dr. Christoph Matheja
Theory of Correct Systems @ UOL
Software Systems Engineering @ DTU

Program Verification – Assignment 3

- Submit your solutions online in groups of 1-3 before the deadline.
- The tasks are numbered consecutively across all assignments. If you have questions, you can thus refer to the task number.
- You can use the concepts and notation from the course material without explanation. Any other definitions or notations must be defined before they are used. Similarly, theorems or mathematical properties that are neither introduced in the course nor well-known from typical undergraduate courses, for example discrete math or logic, need to be introduced and proven first. When in doubt about using a property you know from another course, ask us first.

T9: Error localization (20 Points)

Consider the following program C:

```
{
    assert x == 2;
    assert x > 0
} [] {
    assume x < 2
}
assert x < 3
```

(a) *Annotate the program with* $\mathsf{swp}[C](\emptyset)$ *and determine which assertions should fail.*

(b) *Annotate the program with additional commands to avoid masked verification errors, then compute the* `swp` *and determine which assertions should fail.*

(c) *Encode this program in Viper and check its output with both the Silicon and Carbon backends. Does its output match your expectation?*

T10: Encoding stacks (35 Points)

In the previous assignment you were asked to develop a custom theory for stacks in Z3. We now use Viper to develop an extension of that theory.

(a) *Encode the stack data structure from exercise T5 in Viper by using domains, axiomatizing the functions* `empty`, `push`, `isEmpty`, `pop`, `top`, `sum` *and* `sorted`.

(b) *Prove in Viper the two properties from exercise T5. That is, that the stack*

```
push(top(pop(push(8, push(7, empty)))),
     pop(push(5, push(9, push(12, empty)))))
```

is sorted and has elements that sum up to 28.

(c) *Axiomatize an additional function* `length(s:Stack):Int` *and prove (in Viper) that the following statements hold for* ***arbitrary*** *stacks* s*:*

$$\texttt{length}(s) = 1 \implies \texttt{top}(s) = \texttt{sum}(s)$$

and

$$\texttt{length}(s) = 2 \implies \texttt{top}(\texttt{pop}(s)) = \texttt{sum}(s) - \texttt{top}(s).$$

T11: Encoding match (45 Points)

In the lecture we introduced IVL0, an intermediate verification language that allows the commands `assert` F, `assume` F, command sequencing C_1`;`C_2, as well as non-deterministic choice C_1`[]`C_2. In this task you will extend IVL0 by introducing a new command `match` with the following syntax:

```
match {
   b1 => C1,
   b2 => C2,
   ...
   bn => Cn
}
```

Here, each b_i is a Boolean expression, and each C_i is a command. Informally, this match command will choose which branch to execute depending on the truth value of the guards. You will have to support such a command when building your own verifier in the first project. The goal of this task is thus to lay the groundwork.

(a) *Define operational semantics for the* `match` *command.*

Hint: There are different possible interpretations of how this command should work. We leave the details unspecified on purpose. We will regard your solution as correct as long as your interpretation is reasonable and correctly formalized.

(b) *Develop Hoare-style proof rules for reasoning about the* `match` *command.*

(c) *Show how to encode verification of* `match` *commands into IVL0, and provide an argument as to why your encoding is sound.*

(d) *Provide an example to showcase the semantics, proof rules and encoding of the* `match` *command. You may want to use such examples to test your project 1 implementation.*