Assignment "Correctness Proofs", B3STV 2023/24

June 4, 2024

Deadlines: see website.

Part I (2.5pt)

1. **Refreshing Predicate Logic** [1pt]. Give a formal proof of the formula in the Lecture Notes Section 3.9 no. 7. Use the proof style as in the Lecture Notes.

   Tip: prove the formula via contradiction.

2. **Induction and Equational Proof** [0.8pt] Consider the following two functions to calculate the sum of the elements in a list/sequence. In the notation below, [ ] denotes an empty list, and x:s denotes a list with x as its first element and s as the remaining of the list.

   From programming perspective, these functions can be seen as *programs* that get a list as input and return an integer as output. In fact, you can write them as programs in a functional programming language like Haskell. In programming, the above mentioned [ ] and : act as list constructors.

   $$\begin{array}{lcl} \texttt{sum1}\ [\,] & = & 0 \\ \texttt{sum1}\ (\texttt{x} : \texttt{s}) & = & \texttt{x} + \texttt{sum1}\ \texttt{s} \end{array}$$

   $$\begin{array}{lcl} \texttt{sum2}\ \texttt{n}\ [\,] & = & \texttt{n} \\ \texttt{sum2}\ \texttt{n}\ (\texttt{x} : \texttt{s}) & = & \texttt{sum2}\ (\texttt{x} + \texttt{n})\ \texttt{s} \end{array}$$

   Prove that for all lists of integers `s`, we have:

   $$\texttt{sum1}\ \texttt{s}\ =\ \texttt{sum2}\ 0\ \texttt{s}$$

   To prove a general property about lists, such as the one above, you typically need to use the following list induction rule (quite analogous to the natural number induction rule that you already know).

   $$\frac{P\ [\,] \quad , \quad (\forall x :: (\forall s :: P\ s \Rightarrow P\ (x{:}s)))}{(\forall s :: P\ s)}$$

   Using induction over lists and equational reasoning, to the above claimed equality between `sum1` and `sum2`.

   As a side note, `sum2` has the overhead of having to maintain one extra parameter, but it can be optimized to a more efficient program because it does not actually need to build a call stack.

3. **Weakest pre-condition** [0.4pt]. What is the weakest pre-condition (**wp**) of the following statements with respect to the given post-conditions. All variables are of type `int`.

   Do not just guess the **wp**. Use the **wp**-rules you learned from the lectures to calculate it.

   (a) $\mathtt{x} := \mathtt{x}{+}1;\ \mathtt{y} := \mathtt{x} * \mathtt{y}\ \ \{* \mathtt{x} > 0 \wedge \mathtt{y} > \mathtt{z} *\}$

   (b) **if** x>y
   then x := x−y
   else y := y−x
   $\{* (\exists \mathtt{k} : \mathtt{k} > 0 : \mathtt{d}{*}\mathtt{k} = \mathtt{x}) \wedge (\exists \mathtt{n} : \mathtt{n} > 0 : \mathtt{d}{*}\mathtt{n} = \mathtt{y}) *\}$

4. **Correctness of simple statement** [0.3pt]. Here is again the statement from 2b, this time with a full specification. All variables are of type `int`. To give you a bit intuition: the formula $(\exists \mathtt{k} : \mathtt{k} > 0 : \mathtt{d}{*}\mathtt{k} = \mathtt{x})$ says that `d` is thus a divisor of `x`.

   $\{* \mathtt{d}{>}0 \wedge \mathtt{x} \neq \mathtt{y} \wedge (\exists \mathtt{k} : \mathtt{k} > 0 : \mathtt{d}{*}\mathtt{k} = \mathtt{x}) \wedge (\exists \mathtt{n} : \mathtt{n} > 0 : \mathtt{d}{*}\mathtt{n} = \mathtt{y}) *\}$

   if x>y
   then x := x−y
   else y := y−x

   $\{* (\exists \mathtt{k} : \mathtt{k} > 0 : \mathtt{d}{*}\mathtt{k} = \mathtt{x}) \wedge (\exists \mathtt{n} : \mathtt{n} > 0 : \mathtt{d}{*}\mathtt{n} = \mathtt{y}) *\}$

   Suppose we want to formally prove that the above specification is satisfied. Questions:

   (a) How to use what you learned from 3b?

   (b) Give the *header* of your formal proof. That is, list your assumptions and goal(s). You only need to give the header, not a full proof.

Part II (7.5pt)

1. **Invariant** [0.6pt; 0.3pt each]. Give an invariant for each of the programs below, that would be good enough to prove their corresponding specifications. You do not need to deliver any proof, but do check it on your own scratch that your proposed invariants are indeed good.

   (a)
   $\{* \; i{=}2 \;\wedge\; j{=}20 \; *\}$
   `while` $i{<}j$ `do` $\{\; j{:=}j{-}2 \;;\; i{:=}i{+}1 \;\}$
   $\{* \; j = i \; *\}$

   (b) The program in the Lecture Notes Section 6.14, no. 16.

   Note: the needed invariants are less trivial that you might thought, e.g. for (a) just $\texttt{i} \leq \texttt{j}$ as I is not good enough, while intuitively you might expect it to be an invariant. That intuition is not totally off. However, if you consider the proof obligation IC:

   $$\{* \; \texttt{i}{<}\texttt{j} \wedge \texttt{i}{\leq}\texttt{j} \; *\} \; \texttt{j} := \texttt{j}{-}2 \;;\; \texttt{i} := \texttt{i}{+}1 \; \{* \; \texttt{i}{\leq}\texttt{j} \; *\}$$

   the post-condition of above cannot be proven. Perhaps the problem is that the candidate $\texttt{i} \leq \texttt{j}$ does not contain enough information to close the proof of the post-condition above, and you might want to consider extending it.

2. **Termination metric** [0.4pt]. Consider the following program. It does not do anything exciting, but it still makes an interesting case for you. Give an invariant and a termination metric that would be good enough to prove that the program terminates with the specified post-condition. You do not need to deliver any proof.

   $\{* \; k = 1 \; *\}$
   `while` $k{\neq}10$ `do`
   `if` $k{=}1$ `then` $k{:=}0$ `else` $k{:=}k{+}2$
   $\{* \; k = 10 \; *\}$

   Note that a simple $10 - k$ won't work as your termination metric, as this expression does not always decrease during your iterations.

3. **Proving correctness of a loop** [3.5pt]. The program below checks if `x` occurs as an element of the array-segment `a[0..N)`. The part of the pre-condition that says that the array is sorted is only relevant if you also do the optional-task in question No. 5; otherwise you can drop it.

   $\{* \; \texttt{N}{\geq}\texttt{0} \;\wedge\; \texttt{sorted a N} \; *\}$

   ```
   found := false ;
   i := 0 ;
   while i < N do { found := found ∨ (a[i]=x) ; i:=i+1 }
   ```

   $\{* \;\; ?? \;\; *\}$

   The predicate sorted expresses that the array-segment `a[0..N)` is ascendingly sorted:

   $$\texttt{sorted a N} \;=\; (\forall \texttt{i} : \texttt{0}{\leq}\texttt{i}{<}\texttt{N} : (\forall \texttt{j} : \texttt{i}{\leq}\texttt{j}{<}\texttt{N} : \texttt{a[i]}{\leq}\texttt{a[j]}))$$

Give a formal specification for the program's post-condition, and then give a formal proof of the correctness of the above program (under **total correctness**, so you have to prove its termination as well). You do not need to show the calculation of **wp**.

Additional requirements (also apply for questions No 4 and 5):

- Use the proof style as in the Lecture Notes.
- Clearly justify each proof step with a comment. Your justification is your argument why you consider the step to be a valid proof step; this is just as important as the step itself.
- Proof steps involving quantifications ($\exists$ and $\forall$) should be done in "small" steps (avoid merging such steps into a single one big proof step). A "small" step is defined as a step for which there is a rule or theorem justifying it listed in the Lecture Notes.

  For example, this fragment of an equational proof is good:

  $(\forall \texttt{i} : 0 \leq \texttt{i} < \texttt{k}+1 : \texttt{b}[\texttt{i}])$
  $= \{$ we assumed that $0 \leq k$, then using Domain Merging $\}$
  $(\forall \texttt{i} : 0 \leq \texttt{i} < \texttt{k} \vee \texttt{i} = \texttt{k} : \texttt{b}[\texttt{i}])$
  $= \{ \forall$ Domain Split $\}$
  $(\forall \texttt{i} : 0 \leq \texttt{i} < \texttt{k} : \texttt{b}[\texttt{i}]) \wedge (\forall \texttt{i} : \texttt{i} = \texttt{k} : \texttt{b}[\texttt{i}])$
  $= \{ \forall$ Quantification over Singleton Domain $\}$
  $(\forall \texttt{i} : 0 \leq \texttt{i} < \texttt{k} : \texttt{b}[\texttt{i}]) \wedge \texttt{b}[\texttt{k}]$

  The fragment below, which merges the above steps into one, is **not** allowed in this assignment, since it makes it impossible for your evaluators to determine if you actually understand the underlying formal reasoning:

  $(\forall \texttt{i} : 0 \leq \texttt{i} < \texttt{k}+1 : \texttt{b}[\texttt{i}])$
  $= \{$ we assumed $0 \leq k \}$
  $(\forall \texttt{i} : 0 \leq \texttt{i} < \texttt{k} : \texttt{b}[\texttt{i}]) \wedge \texttt{b}[\texttt{k}]$

4. **A loop with array assignment** [2pt].

   The following program traverses an array `a` from `a[0]` to `a[N-1]`, where $\texttt{N} \geq 1$. If it finds that at a position $\texttt{i} < \texttt{N}-1$ the element `a[i]` is greater than `a[i+1]`, it swaps the two elements. This way, once the loop terminates, no element in `a` positioned at an index less than `N-1` is greater than `a[N-1]`:

   ```
   {* N≥1 *}
   i := 0;
   while ¬ (i = N − 1) do {
     if a[i] > a[i + 1] then {
       tmp:=a[i]; a[i]:=a[i+1]; a[i+1]:=tmp
     }
     else skip;
     i := i+1;
   }
   {* (∀k : 0≤k<N−1 : a[k]≤a[N−1]) *}
   ```

   Provide a formal proof to show that this is a valid specification with respect to a **partial correctness** interpretation. Check the Lecture Notes on how to handle assignments to arrays.

   Some tips. When trying to prove IC, you will have to calculate `wp` *body* Inv. Because the loop's body above involves an if-then-else and array assignments, calculating this `wp` is quite involved. Try to simplify the resulting formula;

this helps later in the proof of IC. Few further tips that might help in this calculation:

- Use the following variant of the wp rule of if-then-else:

  $$\mathsf{wp}\ (\textbf{if}\ g\ \textbf{then}\ S\ \textbf{else}\ T)\ Q \quad = \quad (g \Rightarrow \mathsf{wp}\ S\ Q)\ \wedge\ (\neg g \Rightarrow \mathsf{wp}\ T\ Q)$$

- Section 6.9 of the LN mentions that wp is distributive over conjunction:

  $$\mathsf{wp}\ S\ (Q_1 \wedge Q_2)\ =\ (\mathsf{wp}\ S\ Q_1) \wedge (\mathsf{wp}\ S\ Q_2)$$

  You can use this to split the calculation of wp into several parts for beter overview.

- Calculating wp over an array assignment might generate sub-formulas of the form

  $$a(i\ \textbf{repby}\ e)[k]$$

  By the definition of **repby**, notice that this formula can be rewritten to an if-then-else expression

  $$(k = i) \rightarrow e \mid a[k]$$

  Apply this 'reduction'.
  Related to that, the following lemma might be useful for you (you have to prove it first, if you use it):

  Lemma: for $0 \leq i$ we have:
  $\mathsf{wp}\ (a[i] := a[i+1]; a[i+1] := tmp)\ (\forall k : 0 \leq k < i{+}1 : a[k] \leq a[i+1])$
  $=$
  $(\forall k : 0 \leq k < i : a[k] \leq tmp)\ \wedge\ a[i+1] \leq tmp)$

- There are theorems in Appendix A for rewriting if-then-else expressions. E.g. you can rewrite this:
  $(cond \rightarrow x \mid y) \leq 0$
  $=$ // Theorem A.4.2
  $cond \rightarrow\ x \leq 0 \mid y \leq 0$
  $=$ // Theorem A.4.5 (COND-split)
  $(cond \Rightarrow x \leq 0)\ \wedge\ (\neg cond \Rightarrow y \leq 0)$

5. **Optional: Loop with breaks** [1pt]. Here is a more efficient variation of the program in No. 3. The variant below will 'break' the loop right after `x` is found, or when the current `a[i]` exceeds `x` (well, `a` is sorted, so searching further would then be vain).

   $\{*$ `N≥0` $\wedge$ `sorted a N` $*\}$

   ```
   found := false ;
   i := 0 ;
   while i<N ∧ ¬found ∧ a[i]≤x do {
       found := found ∨ (a[i]=x) ;
       i:=i+1
   }
   ```

   $\{*$ same post-cond as in the simple variant in No. 3 $*\}$

   Formally prove the correctness of the above program. Since in No. 3 you already proved the correctness of the 'simple' variant that has no break, for the above program you only need to redo your PEC (Proof of the Exit Condition).

   Additional Requirements: the same as in No. 3.