McMaster University
Department of Computing and Software
Musa Al-hassy

COMP SCI 3EA3
Quiz 5
Solution Hints

COMP SCI 3EA3 — Software Specification and Correctness

March 22, 2017

Name _____ Student Number _____

Special Instructions:

- This examination paper includes 6 pages (including this cover page) and 3 questions.
  You are responsible for ensuring that your copy of the paper is complete. Bring any discrepancy to the attention of your invigilator.
- **Read each question completely and carefully** before answering it.
- Answer all questions.
- **In doubt, document!**

All the questions are closely related to or directly from the lectures and sheets!

Contents

1 *Sqrt* Forevermore — 9 marks —

For $N : \mathbb{Z}$, solve for ? in the following triple such that your solution is **logarithmic** in time,

$$\{1 \le N\} \;\; ? \;\; \{x^2 \le N < (x+1)^2\}$$

—remember that a such a triple is a theorem and so requires a PROOF!

You *may* need to use the facts that 0 is a fixed point of the squaring function and the squaring function is strictly monotonic on $\mathbb{N}$.

Solution Hints:

The hint 'logarithmic' is not-so-subtly suggestive of using "Binary Search" —the algorithm schema covered over the past week and a half and the only one referencing to logarithm time on the theorems sheet.

Comparing the desired postcondition '$x^2 \le N < (x+1)^2$ with that of Binary Search, it seems best if we take $\mathcal{Z}$ to be such that

$$x \,\mathcal{Z}\, y \quad \equiv \quad x^2 \le N < y^2$$

To apply Binary Search, we must confirm the proviso: Our relation is co-transitive. However, for the integers we have $N < y^2 \;\equiv\; \neg(y^2 \le N)$ and so our relation can be rewritten as $x \,\mathcal{Z}\, y \quad \equiv \quad x^2 \le N \;\wedge\; \neg(y^2 \le N)$ and this relation has already been proven to be co-transitive —with incremental proof steps carried out upto exercise 9.0.7 of Sheet 9.

With that out of the way, we have:

$\{\, 1 \le N \,\}$
skip
; $\{\, 0 < N \wedge 0^2 \le N < (N+1)^2 \,\}$ ..... skip-rule with properties of $-^2$
$x, y := 0, N$
; $\{$ Invariant $0 \le x \;<\; y \;\le\; N \wedge x^2 \le N < y^2$, Bound $y \;-\; x$ $\}$
do $x + 1 \neq y \;\rightarrow$
  $m := (x + y) \div 2$
  $\{x < m < y\}$
  ; **if** $m^2 \le N < y^2 \;\rightarrow\; x := m$
  ▯ $x^2 \le N < m^2 \;\rightarrow\; y := m$
  fi
od
; $\{\, 0 \le x < N \;\wedge\; x^2 \le N < (x+1)^2 \,\}$ ..... Binary Search
skip
$\{\, x^2 \le N < (x+1)^2 \,\}$ ..... skip-rule with Weakening

We clean up this proof outline by only keeping the first and last assertions, then we use the fact that skip is the unit of sequencing to remove those and finally use the invariant to remove the redundancies in the conditional's guards; resulting in:

$\{\, 1 \le N \,\}$
$x, y := 0, N$
; **do** $x + 1 \neq y \;\rightarrow$
  $m := (x + y) \div 2$
  ; **if** $m^2 \le N \;\rightarrow\; x := m$
  ▯ $N < m^2 \;\rightarrow\; y := m$
  fi
od
$\{\, x^2 \le N < (x+1)^2 \,\}$

Observe that a more quick solution would have been to use the predicate version of Binary Search of the Theorem's List with $b\, m, N := m^2 \le N, N + 1$ and it has no proviso; then weakening its post-condition to the one desired above. We took a longer route so that there is another example of instantiating the general Binary Search. Besides being covered in tutorial, Quiz 4 and the lecture of March 15th both suggested this exact problem.

2 Binary Search Bonanza — 21 marks —

With reference to the following algorithms, choose **ONLY** seven (7) of the following TRUE/FALSE questions to answer **and** provide justifications to your answers. Answers with no justifications receive **zero marks**, and good justifications get a non-zero mark irrespective of whether the right checkbox is ticked or not.

If more than 7 questions are answered, only the first seven encountered will be considered.

(Predicate Instance of Binary Search)

Provided predicate b defined on $0..N-1$,

$\{\, 0 \le N \wedge b\,(-1) \wedge \neg b\, N \,\}$
$x, y := -1, N$
; **do** $x + 1 \neq y \rightarrow$
  $m := (x + y) \div 2$
  ; **if** $b\, m \rightarrow x := m$
  ▯ $\neg b\, m \rightarrow y := m$
  fi
od
$\{\, -1 \le x < N \wedge b\, x \wedge \neg b\,(x+1) \,\}$

(General Binary Search)

Provided $\mathcal{Z}$ is a co-transitive relation,

$\{\, a < b \wedge a \mathcal{Z}\, b \,\}$
$x, y := a, b$
; $\{$ Invariant $a \le x < y \le b \wedge x \mathcal{Z}\, y$, Bound $y - x \,\}$
do $x + 1 \neq y \rightarrow$
  $m := (x+y) \div 2$
  ; **if** $m \mathcal{Z}\, y \rightarrow x := m$
  ▯ $x \mathcal{Z}\, m \rightarrow y := m$
  fi
od
$\{\, a \le x < b \wedge x \mathcal{Z}\,(x+1) \,\}$

TRUE FALSE

1. ☑ □ In the predicate variant, the algorithm requires at most $\log_2(N+1)$ repetitions.

   Solution Hints:
   Although the bound function is $y-x$, each repetition halves $y-x$, and so a tighter bound function is $\log_2(y-x)$, which is decreased by 1 each repetition. Since initially $x, y := -1, N$, it requires $\log_2(N+1)$ iterations.

2. □ ☑ Binary Search is fast since it separates the search space into 2 pieces; an improvement, in the antitonic case, would be to split the search space into more pieces and the resulting algorithm will be slightly more complicated but far more efficient!

   Solution Hints:
   If we split the space into k pieces then, due to antitonicity, we have to make at most k comparisons to figure out which piece we should look at for the next iteration and so the algorithm is $\mathcal{O}(k \times \log_k N)$ and this is the same as $\mathcal{O}(\log_2 N)$ since multiplicative constants don't really matter for large inputs. Whence, a more complicated algorithm that is essentially as efficient as Binary Search.

3. □ ☑ Binary Search is best used if the underlying array b is sorted.

   Solution Hints:
   Binary search makes no mention of monotonicity; it's purpose is to produce an index x satisfying b such that its neighbour $x+1$ is not a solution to b.
   In the special case that b is antitonic, then the algorithm ensures the resulting index is the largest solution to b. However, the this is a rewriting of the post-condition and not of the program internals, so there is no improvement to the algorithm.

4. ☑ □ In the general schema, the co-transitivity condition ensures that the conditional is well-defined and so does not abort.

   Solution Hints:
   That the disjunction of the guards is true follows from the invariant, is immediate from the relation being co-transitive.

5. ☑ ☐ If you were to play the "guess my number game" in the interval 1 to 1 billion and you guessed each number in sequence —ie Linear Search— then it would take you at most 1 billion guesses, whereas if you guessed using a Binary Search approach it would take you about 30 tries. If the interval was enlarged upto 4 billion, the linear search approach would worsen for each new element thereby taking at most 4 billion guesses, whereas Binary Search barely grows; requiring 2 more tries for the additional 3 billion new elements!

Solution Hints:

The computer-lovers 'kilo' is 1024 or 2^{10}, so by the power rule for logarithms we have $\log_2(1\text{ billion}) = \log_2 10^9 = \log_2(10^3)^3 = 3 \cdot \log_2 1000 \leq 3 \cdot \log_2 1024 = 3 * 10$ —more accurately $\log_2(1\text{ billion}) \approx 29.897 \approx 30$.

Similarly, by the multiplication-to-addition rule of logs, $\log_2(4\text{ billion}) = \log_2 4 + \log_2(1\text{ billion}) \approx 2 + 30$.

Alternatively: Binary Search splits the search space in half at each iteration and so the latter problem of 4 billion elements can be simplified to the former problem via two extra iterations since: $(4\text{ billion} \div 2) \div 2 = 1\text{ billion}$.

Of-course on a machine, your integer sizes are limited!

(This problem is exercise 10.1.7 of Sheet 10.)

6. ☐ ☑ Some iterations do not make progress, that is do not decrease the bound since the midpoint of two numbers is not always strictly between both numbers; for example the integral midpoint of 1 and 2 is 1.

Solution Hints:
The question is misleading since the midpoint is calculated in the context $x + 2 \leq y$ and so one can prove that $x < (x+y) \div 2 < y$ and one of x or y is assigned the midpoint and so $x - y$, the bound, is decreased each iteration.
The given scenario of $x, y = 1, 2$ is not possible in the loop body since the loop guard forbids it:
$1 + 1 \neq 2 \equiv \mathit{false}$!

7. ☑ ☐ Every co-transitive relation is *precisely* the complement of a retract of a transitive relation; ie $\mathcal{Z}$ is co-transitive iff it is of the form $\neg(f\ x \sim f\ y)$ for a transitive relation $\sim$.

Solution Hints:
The forwards direction is obtained by taking $\sim$ to be the complement of $\mathcal{Z}$ and $f = \mathsf{Id}$, while the converse direction follows immediately from the transitivity of $\sim$ and is proved in exercise 9.0.6 of Sheet 9.

8. ☑ ☐ For antitonic b, if $\neg b\,0$ holds initially in the predicate variant of Binary Search, the algorithm will establish $x = -1$.

Solution Hints:
Antitonicity means once we see a FALSE then everything after will also be a FALSE. So $b(-1)$ is true and its neighbour $b\,0$ is false; moreover these are the only true-false neighbours since all other elements $1..N$ are false, by antitocity. Hence, the algorithm must return $x = -1$.
Alternatively: By antitonicity and $\neg b\,0$, we have that the entire array is false and so every check $b\,m$ yields false. So y decreases at each iteration whereas x is never altered and maintains its initial value of -1.
Alternatively, this is one-point rule for antitone body that I made up for the theorem's list.
(This problem is exercise 10.0.8 of Sheet 10.)

9. ☑ ☐ If $b\,0$ holds initially in the predicate variant of Binary Search, the algorithm will establish $x \geq 0$ —thus we can replace $x := -1$ with $x := 0$ in the beginning of the algorithm and strengthen the post-condition by adding $x \geq 0$.

Solution Hints:
If $b\,0$ is true, then x cannot be -1 since its neighbour, which is $b\,0$, is not false and so it would not satisfy the post-condition. The post-condition ensures that $-1 \leq x < N$ but we've just argued that $x \neq -1$, and so the post-condition yields $0 \leq x$.

Notice that there is no mention of order properties on b!

10. ☐ ☑ Suppose we use the predicate variant of Binary Search on the predicate $b : \mathbb{Z} \to \mathbb{B} : m \mapsto \mathit{true}$ on the interval $0..N-1$ with $N = 10$. Since b is antitonic, the algorithm returns the largest index in $0..N-1$ satisfying b and so it ensures $x = N-1$. Moreover it ensures $b(N) \wedge \neg\, b(N+1)$ and so any immediate code after the algorithm may use the fact that $b(11)$ is false.

Solution Hints:
No, since b at -1 and N is fictious —imagined!
That is, $b(-1)$ and $b\,N$ are thought variables to make the logical reasoning go through and that if bN is actually defined, it may not satisfy the logical assertions of the fictitious bN needed to make the algorithm's current presentation.

11. ☑ ☐ Computing the midpoint using $m := x + (y - x) \div 2$ is not *always* better than $m := (x + y) \div 2$.

Solution Hints:
This problem is exercise 10.0.6 of Sheet 10.

The usual argument in favour of the alternate form is to avoid overflow. Yet, what if we are looking at a sequence —or our program allows negative sub-scripting— and so $x < 0$ is possible, then $-x > 0$ and so $y - x$ could still result in overflow.

Thus, such an algebraic alteration is not guaranteed to fix the issue for non-conventional array indexing —such as evaluating a given lambda, function-object, or using non-natural number indices for array indexing. This idea is also known under the name of "generic indexing" and has a variety of useful applications. We've been using it in class thus-far to avoid undefined elements.
(Some imperative languages permit overloading the array-subscripting operation $-[-]$ as well.)

12. ☐ ☑ If b is antitonic, then the algorithm returns the smallest solution to b.

Solution Hints:
No, the theorems list clearly indicates —due to local characterisation of integer extremea— that for antitonic b, the algorithm computes the *largest* solution.

13. ☑ ☐ If b is antitonic, we can replace $y := m$ with $y := m - 1$ in the predicate variant of the algorithm.

Solution Hints:
The invariant yields that $x..y$ is the sub-interval containing the greatest solution.

If the middle of the interval containing our solution is FALSE, then —since b is antitonic— everything after it will also be FALSE and so we can discard the second half of the interval since we're looking for a solution. Moreover, we can discard the midpoint as well since it is not a solution, but do not do so in to avoid breaking the symmetry with the first case there-by making the algorithm more complicated to understand, reason about, and remember.

More importantly, discarding the midpoint would invalidate our invariant and as such we would necessarily need a new —possibly more complicated— invariant that allows such a command; possibly along with a new loop guard.

More concretely, if we DO NOT ALTER the loop guard nor the invariant then we can loop forever with the counterexample being the singleton $b = [\mathit{False}]$. Here $N = 1$ and so x, y are initialised to $-1, 1$ and thus $x + 1 = -1 + 1 = 0 \neq 1 = y$ and we enter the loop to obtain $m = (x + y) \div 2 = 0$ and since $\neg b\,0$ we NOW decrease y to $m - 1$, ie -1 and then loop again since $x = y = -1$; now the midpoint is $(-1 + -1) \div 2 = -1$ and so the variables values never differ and we loop indefinitely. With this however, the answer of FALSE is also acceptable —hence the justifications!

(This problem follows from exercise 10.0.5 of Sheet 10.)

14. ☑ ☐ The above general schema is sufficiently symmetric and as such easy to remember and derive.

Solution Hints:
I believe it to be simple but you may not and that's fine too; we can talk about it to see each other's perspectives.

3 (Bonus) The Return of The Great De Morgan — 5 marks —

Solution Hints:

In this bonus exercise, we'd like to prove the following properties:

- The union of two class rooms is empty *precisely* when the rooms are themselves empty.
- The greatest pay a union of workers makes is minimum wage *precisely* when all its members make minimum wage.
- The catenation of two sequences is empty *precisely* when the given sequences are themselves empty.
- The least tree containing two given trees is empty *precisely* when the given trees themselves are empty.

Rather than prove each of these results directly, we realise them as the same result in the abstract setting of bounded join-lattices. That is, for all x and y, we calculate:

Proving $x \sqcup y = \bot \quad \equiv \quad x = \bot \quad \wedge \quad y = \bot$ **:**

$$
\begin{array}{ll}
 & x \sqcup y = \bot \\
= & \quad \{ \quad \text{Antisymmetry} \quad \} \\
 & x \sqcup y \sqsubseteq \bot \ \wedge \ \bot \sqsubseteq x \sqcup y \\
= & \quad \{ \quad \text{Bottom Element and identity of } \wedge \quad \} \\
 & x \sqcup y \sqsubseteq \bot \\
= & \quad \{ \quad \sqcup\text{-characterisation} \quad \} \\
 & x \sqsubseteq \bot \quad \wedge \quad y \sqsubseteq \bot \\
= & \quad \{ \quad \text{Bottom Element and identity of } \wedge \quad \} \\
 & (x \sqsubseteq \bot \ \wedge \ \bot \sqsubseteq x) \quad \wedge \quad (y \sqsubseteq \bot \ \wedge \ \bot \sqsubseteq y) \\
= & \quad \{ \quad \text{Antisymmetry} \quad \} \\
 & x = \bot \quad \wedge \quad y = \bot
\end{array}
$$