CSE 3311 Software Design
Report 2 Partial Solution

Expect diagrams throughout. Only a few diagrams are given for illustration. Diagrams must be labeled and referenced in the body of the report. The partial solution does not contain sufficient commentary about the assertions. Your solution is expected to have more explanation and justification.

1 Greatest common divisor (GCD) contract

require
a_strickly_positive: $a > 0$
b_strickly_positive: $b > 0$

invariant
remainder_big_enough: $remainder \geq 0$
remainder_small_enough: $remainder < y$
x_stricly_positive: $x > 0$
y_stricly_positive: $y > 0$
gcd_x_y_related_to_gcd_a_b: $gcd(a, b) = gcd(x, y)$

variant
remainder

ensure
result_strictly_positive: $Result > 0$
result_divides_both: $(a \bmod Result = 0) \wedge (b \bmod Result = 0)$
result_is_greatest: $\forall\, x : Result + 1 .. \min(a, b) \bullet (a \bmod x \neq 0) \wedge (b \bmod x \neq 0)$

2 Cumulative sum contract

require

in_exists: in $\neq$ void
in_proper_lower_bound: in.lower = 1
n_in_range: in.lower $\leq$ n $\leq$ in.upper

invariant

partial_result_correct: $\forall$ p : 1 .. j • Result[p] = $\sum_{k=1}^{p}$ in[k]

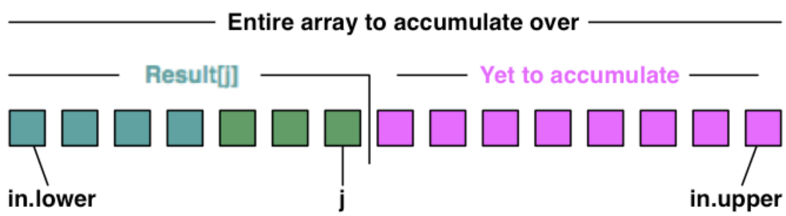

Figure 1: Diagram showing invariant for cumulative sum loop

variant

n – j

ensure

result_proper_size: Result.lower = 1 $\wedge$ Result.upper = n

result_correct: $\forall$ j : 1 .. n • Result[j] = $\sum_{k=1}^{j}$ in[k]

3 Separate even-odd contract

require

in_exists: in $\neq$ void

invariant

partialresult_correct:

$\forall j : \text{in.lower} .. \text{max\_even} \bullet \text{even}(\text{in}[j])$

$\forall j : \text{min\_odd} .. \text{in.upper} \bullet \text{odd}(\text{in}[j])$

known_evens_in_original:

$\forall j : \text{in.lower} .. \text{max\_even} \bullet \text{in}'[j] \in \{k : \text{in.lower} .. \text{in.upper} \bullet \text{in}[k]\}$

known_odds_in_original:

$\forall j : \text{min\_odd} .. \text{in.upper} \bullet \text{in}'[j] \in \{k : \text{in.lower} .. \text{in.upper} \bullet \text{in}[k]\}$

original_in_result:

$\forall j : \text{in.lower} .. \text{max\_even} \bullet \text{even}(\text{in}[j]) \rightarrow \text{in}[j] \in \{k : \text{in.lower} .. \text{max\_even} \bullet \text{in}'[k]\}$

$\forall j : \text{min\_odd} .. \text{in.upper} \bullet \text{even}(\text{in}[j]) \rightarrow \text{in}[j] \in \{k : \text{in.lower} .. \text{max\_even} \bullet \text{in}'[k]\}$

$\forall j : \text{in.lower} .. \text{max\_even} \bullet \text{odd}(\text{in}[j]) \rightarrow \text{in}[j] \in \{k : \text{min\_odd} .. \text{in.upper} \bullet \text{in}'[k]\}$

$\forall j : \text{min\_odd} .. \text{in.upper} \bullet \text{odd}(\text{in}[j]) \rightarrow \text{in}[j] \in \{k : \text{min\_odd} .. \text{in.upper} \bullet \text{in}'[k]\}$

variant

min_odd – max_even

ensure

result_correct: $\forall j : \text{in.lower} .. \text{Result}-1 \bullet \text{even}(\text{in}'[j])$

$\forall j : \text{Result} .. \text{in.upper} \bullet \text{odd}(\text{in}'[j])$

results_in_original:

$\forall j : \text{in.lower} .. \text{in.upper} \bullet \text{in}'[j] \in \{k : \text{in.lower} .. \text{in.upper} \bullet \text{in}[k]\}$

original_in_results:

$\forall j : \text{in.lower} .. \text{in.upper} \bullet \text{in}[j] \in \{k : \text{in.lower} .. \text{in.upper} \bullet \text{in}'[k]\}$

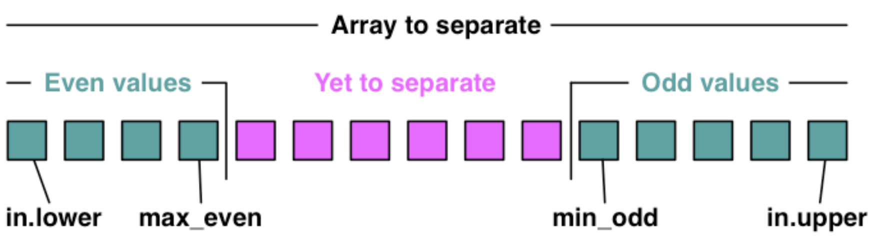

Figure 2: Diagram for loop invariant

The ensure clause is only partially correct. Actually need to specify that the number of occurrences of each value in the original array occur in the final array. Here we regard the array as a sequence; i.e. the indices map to the values. The operator ⌈ is the sequence restriction operator. It extracts, for example in the first clause, the subsequence from the sequence in that corresponds to values in the set {in[j]}. Because the set {in[j]} contains only one value, the expression {k : in.lower .. in.upper • in ⌈ {in[j]} } creates a set of sequences with one sequence for each value in the array in. If the before and after set of sequences is the same, then we have neither gained nor lost copies of a value if it occurs multiple times. This clause is difficult for you to write mathematically but it is not unreasonable for you to have thought of and to describe the problem.

original_and_results_have_the_same_values:

$\forall j : \text{in.lower} .. \text{in.upper} \bullet \{ \text{in} \lceil \{\text{in}[j]\} \} = \{ \text{in}' \lceil \{\text{in}'[j]\} \}$

4 Verify double_half algorithm is correct

Question 0: What is the loop invariant?

We are given the loop invariant

$$\forall\ j : in.lower\ ..\ k{-}1 \mid odd(in[j]) \bullet in'[j] = in[j]*2$$
$$\wedge\ \forall\ j : in.lower\ ..\ k{-}1 \mid even(in[j]) \bullet in'[j] = in[j]/2$$

Question 1: Is the base case established?

From the **from** clause of the loop statement the following relationships are true.

$$k = in.lower$$

Substitute into the loop invariant to get the following.

$$\forall\ j : in.lower\ ..\ in.lower{-}1 \mid odd(in[j]) \bullet in'[j] = in[j]*2$$
$$\wedge\ \forall\ j : in.lower\ ..\ in.lower{-}1 \mid even(in[j]) \bullet in'[j] = in[j]/2$$

In both clauses the interval is empty so the predicate is true. As a consequence the invariant is true.

Question 2: Verify inductive case

We assume the invariant is true

$$\forall\ j : in.lower\ ..\ k{-}1 \mid odd(in[j]) \bullet in'[j] = in[j]*2$$
$$\wedge\ \forall\ j : in.lower\ ..\ k{-}1 \mid even(in[j]) \bullet in'[j] = in[j]/2$$

Executing the body of the loop gives two cases to consider.

Case 1: even(in[k])

Executing the loop body gives the following relationships

$$k' = k + 1$$
$$\wedge\ in'[k] = in[k]/2$$

The loop invariant at the end of the loop is the following.

$$\forall\ j : in.lower\ ..\ k'{-}1 \mid odd(in[j]) \bullet in'[j] = in[j]*2$$
$$\wedge\ \forall\ j : in.lower\ ..\ k'{-}1 \mid even(in[j]) \bullet in'[j] = in[j]/2$$

Substitute $k' = k + 1$ into the loop invariant.

$$\forall\ j : in.lower\ ..\ k \mid odd(in[j]) \bullet in'[j] = in[j]*2$$
$$\wedge\ \forall\ j : in.lower\ ..\ k \mid even(in[j]) \bullet in'[j] = in[j]/2$$

Split off the last term in each range.

$$\forall\ j : in.lower\ ..\ k{-}1 \mid odd(in[j]) \bullet in'[j] = in[j]*2 \wedge odd(in[k]) \rightarrow in'[k] = in[k]*2$$
$$\wedge\ \forall\ j : in.lower\ ..\ k{-}1 \mid even(in[j]) \bullet in'[j] = in[j]/2 \wedge even(in[k]) \rightarrow in'[k] = in[k]/2$$

In the first line, the first clause is true because it is the same as in the loop invariant at beginning of the loop. The second clause is true because odd(in[k]) is false, so the implication is true. As a consequence the first line of the invariant is true.

In the second line, the first clause is true because it is the same as in the loop invariant at the beginning of the loop. The second clause is true because even([in[j]) is true and in'[k] = ink]/2 is true, so the implication is true.

As a consequence the loop invariant is true at the end of execution the loop body for this case.

Case 2: odd(in[k])

Executing the loop body gives the following relationships

$$k' = k + 1$$
$$\wedge\ in'[k] = in[k]*2$$

The loop invariant at the end of the loop is the following.

$$\forall\ j : in.lower\ ..\ k'{-}1 \mid odd(in[j]) \bullet in'[j] = in[j]*2$$

$\wedge\ \forall\ j : in.lower .. k'–1 \mid even(in[j]) \bullet in'[j] = in[j]/2$

Substitute $k' = k + 1$ into the loop invariant.

$\forall\ j : in.lower .. k \mid odd(in[j]) \bullet in'[j] = in[j]*2$
$\wedge\ \forall\ j : in.lower .. k \mid even(in[j]) \bullet in'[j] = in[j]/2$

Split off the last term in each range.

$\forall\ j : in.lower .. k–1 \mid odd(in[j]) \bullet in'[j] = in[j]*2 \wedge odd(in[k]) \rightarrow in'[k] = in[k]*2$
$\wedge\ \forall\ j : in.lower .. k–1 \mid even(in[j]) \bullet in'[j] = in[j]/2 \wedge even(in[k]) \rightarrow in'[k] = in[k]/2$

In the first line, the first clause is true because it is the same as in the loop invariant at beginning of the loop. The second clause is true because odd([in[j]) is true and in'[k] = ink]*2 is true, so the implication is true.

In the second line, the first clause is true because it is the same as in the loop invariant at the beginning of the loop. The second clause is true because even(in[k]) is false, so the implication is true. As a consequence the second line of the invariant is true.

As a consequence the loop invariant is true at the end of execution the loop body for this case.

Since the loop invariant remains true in both cases, the executing the loop body preserves the loop invariant.

Question 3a: Does the loop terminate?

The termination condition is $k > in.upper$.
k starts at in.lower and increases by 1 on every iteration of the loop. Eventually k must become greater than in.upper. As a consequence the loop terminates.

Question 3b: Is the postcondition established?

The loop invariant is the following.

$\forall\ j : in.lower .. k–1 \mid odd(in[j]) \bullet in'[j] = in[j]*2$
$\wedge\ \forall\ j : in.lower .. k–1 \mid even(in[j]) \bullet in'[j] = in[j]/2$

At the end of the loop $k > in.upper$ and k increments by 1, therefore $k = in.upper + 1$.

Substitute into the loop invariant to get the following.

$\forall\ j : in.lower .. (in.upper+1)–1 \mid odd(in[j]) \bullet in'[j] = in[j]*2$
$\wedge\ \forall\ j : in.lower .. (in.upper+1)–1 \mid even(in[j]) \bullet in'[j] = in[j]/2$

Which simplifies to the following.

$\forall\ j : in.lower .. in.upper \mid odd(in[j]) \bullet in'[j] = in[j]*2$
$\wedge\ \forall\ j : in.lower .. in.upper \mid even(in[j]) \bullet in'[j] = in[j]/2$

The last expression is the same as the post condition. As a consequence the postcondition is true.

Q.E.D.