CS 310, Assignment 7

Answers

1. Use the array-component assignment axiom two times to find the (most general) sufficient pre-condition P for the following code fragment:

   ```
   ASSERT( P ) /* determine what is P */
   A[i] = x;
   A[j] = 2;
   ASSERT( A[k] >= x )
   ```

   Assume that `x` is an integer variable, `A` is an array of integers, and all the subscripts are within the range of subscripts for `A`.

   Write first the assertion `P` using the notation from the array-component assignment axiom, and then rewrite `P` in a logically equivalent and simplified form that does not contain any notation $(\mathtt{A}|\mathtt{I} \mapsto \mathtt{E})$.

   ANSWER: We construct the following tableau from bottom to top:

   $\mathtt{ASSERT}((\mathtt{k} == \mathtt{j}\ \&\&\ 2 >= \mathtt{x})\ ||\ (\mathtt{k}! = \mathtt{j}\ \&\&\ \mathtt{k} == \mathtt{i})\ ||\ (\mathtt{k}! = \mathtt{j}\ \&\&\ \mathtt{k}! = \mathtt{i}\ \&\&\ \mathtt{A}[\mathtt{k}] >= \mathtt{x}))$
   $\mathtt{ASSERT}(((\mathtt{A}|\mathtt{i} \mapsto \mathtt{x})|\mathtt{j} \mapsto 2)[\mathtt{k}] >= \mathtt{x})$
   `A[i] = x;`
   $\mathtt{ASSERT}((\mathtt{A}|\mathtt{j} \mapsto 2)[\mathtt{k}] >= \mathtt{x})$
   `A[j] = 2;`
   $\mathtt{ASSERT}(\mathtt{A}[\mathtt{k}] >= \mathtt{x})$

   The last (top) pre-condition strengthening eliminates the array substitutions and happens as follows:

   - Whenever `k == j` we have `A[k] == 2`.
   - If `k != j` but `k == i` we have that `A[k] == A[i] == x` and so the condition `A[k] >= x` becomes `x == x` which is equivalent to `true`.
   - In all other cases the post-condition remains unchanged.

   ---

2. Assume a declarative interface where `n` and `max` are constant integers, and `A` is an array of integers of size `max`. Consider the following correctness statement:

```
ASSERT(1 < n <= max)
int i;
i=2;
A[0]=1;
A[1]=1;
while (i < n ) {
    A[i] = A[i-1] + A[i-2];
    i++;
}
ASSERT(Forall (k=2; k < n) A[k] == A[k-1] + A[k-2])
```

(a) Give a complete proof tableau for the above correctness statement by adding all the intermediate assertions. State all the mathematical facts that are used in the proof.

Hint: Recall our discussion in class that the following two assertions are obvious tautologies:

```
Forall (k=i; k<j) P(k)   ==   P(i) && Forall (k=i+1; k<j) P(k)
Forall (k=i; k<j) P(k)   ==   P(j-1) && Forall (k=i+1; k<j-1) P(k)
```

That is, we can always separate as an extra conjunctive term an "end" of the range in a universally quantified property.

ANSWER:

```
ASSERT(1 < n <= max)
int i;
ASSERT(1 < n <= max)
FACT((k=2; k < 2) is an empty range)
ASSERT(1 < 2 < n <= max &&
       Forall (k=2; k < 2) ((A|0->1)|1->1)[k] ==
                           ((A|0->1)|1->1)[k-1] + ((A|0->1)|1->1)[k-2])
i=2;
ASSERT(1 < i <= n <= max &&
       Forall (k=2; k < i) ((A|0->1)|1->1)[k] ==
                           ((A|0->1)|1->1)[k-1] + ((A|0->1)|1->1)[k-2])
A[0]=1;
ASSERT(1 < i <= n <= max &&
       Forall (k=2; k < i) (A|1->1)[k] == (A|1->1)[k-1] + (A|1->1)[k-2])
A[1]=1;
ASSERT(1 < i <= n <= max && Forall (k=2; k < i) A[k] == A[k-1] + A[k-2])
while (i < n ) {
    ASSERT(1 < i <= n <= max && Forall (k=2; k < i) A[k] == A[k-1] + A[k-2]
           && i < n)
    FACT(i < n => i+1 <= n)
    ASSERT(1 < i+1 <= n <= max && Forall (k=2; k < i) A[k] == A[k-1] + A[k-2]
           && i < n)
    ASSERT(1 < i+1 <= n <= max && Forall (k=2; k < i) A[k] == A[k-1] + A[k-2])
```

```
    ASSERT(1 < i+1 <= n <= max && Forall (k=2; k < i)
            A[k] == A[k-1] + A[k-2] && A[i-1] + A[i-2] == A[i-1] + A[i-2])
    FACT((A|i->A[i-1] + A[i-2])[i] == A[i-1] + A[i-2] &&
         (A|i->A[i-1] + A[i-2])[i-1] == A[i-1] &&
         (A|i->A[i-1] + A[i-2])[i-2] == A[i-2])
    ASSERT(1 < i+1 <= n <= max &&
           Forall (k=2; k < i)
              A[k] == A[k-1] + A[k-2] &&
              (A|i->A[i-1] + A[i-2])[i] ==
              (A|i->A[i-1] + A[i-2])[i-1] + (A|i->A[i-1] + A[i-2])[i-2])
    FACT(Forall (k=2; k < i+1) P(k) == Forall (k=2; k < i) P(k) && P(i))
    ASSERT(1 < i+1 <= n <= max && Forall (k=2; k < i+1)
                     (A|i->A[i-1] + A[i-2])[k] ==
                        (A|i->A[i-1] + A[i-2])[k-1] +
                        (A|i->A[i-1] + A[i-2])[k-2])
    A[i] = A[i-1] + A[i-2];
    ASSERT(1 < i+1 <= n <= max &&
           Forall (k=2; k < i+1) A[k] == A[k-1] + A[k-2])
    i++;
    ASSERT(1 < i <= n <= max &&
           Forall (k=2; k < i) A[k] == A[k-1] + A[k-2])
}
ASSERT(1 < i <= n <= max && i >= n &&
       Forall (k=2; k < i) A[k] == A[k-1] + A[k-2])
ASSERT(1 < i <= n <= max && i >= n &&
       Forall (k=2; k < n) A[k] == A[k-1] + A[k-2])
ASSERT(Forall (k=2; k < n) A[k] == A[k-1] + A[k-2])
```

(b) Provide a formal argument for the total correctness of the statement.

ANSWER: We note that `n-i` is a variant for the loop. Indeed, `i<=n` is part of the invariant and so `n-i>=0`. Then `n-i` decreases monotonically and so it will reach the value `0`. Finally, `n-i==0` implies that `i<n` is false and so the loop terminates. We thus conclude that the program always terminates.

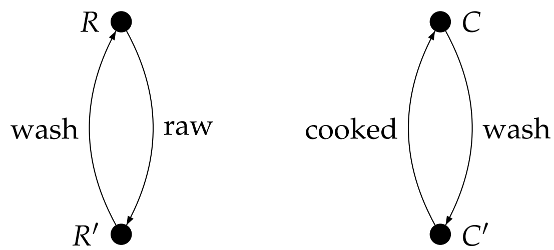
---

3. Consider the following specification of a shop handling raw and cooked meat. The actions of handling raw meat, handling cooked meat and washing hands are represented by the CSP actions raw, cooked, and wash, respectively.

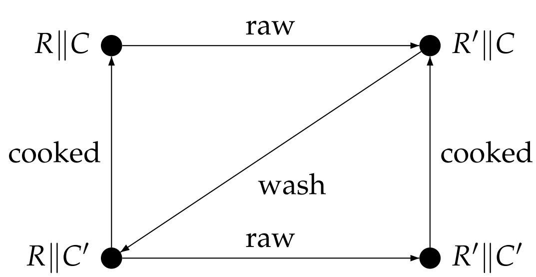
$$
\begin{array}{lcl}
\text{RAW} & = & \text{raw} \rightarrow \text{wash} \rightarrow \text{RAW} \\
\text{COOKED} & = & \text{wash} \rightarrow \text{cooked} \rightarrow \text{COOKED} \\
\text{SHOP} & = & \text{RAW} \mid\mid \text{COOKED}
\end{array}
$$

(a) Give the transition graph (or finite automaton) of the process SHOP.

ANSWER: The transition graphs for RAW and COOKED (abbreviated R and C, respectively) are as follows:



The only synchronized action is wash (common between the two processes) while raw and cooked are unsynchronized. We then have the following transition graph for SHOP:



(b) A common hygiene requirement while handling meat is that hands must be washed between handling raw and cooked meat. Does SHOP meet this requirement? Explain why or why not.

ANSWER: Clearly the hygiene requirement is not observed. After the first raw event the hands are washed, but thereafter either raw meat is handled immediately after cooked meat (following the path $R\|C' \xrightarrow{\text{cooked}} R\|C \xrightarrow{\text{raw}} R'\|C$) or cooked meat is handled immediately after raw (along the path $R\|C' \xrightarrow{\text{raw}} R'\|C' \xrightarrow{\text{cooked}} R'\|C$).
