CS 310, Assignment 7

Due on 8 April in class

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

```
ASSERT(0 <= n <= max)
int i;
i = n;
A[i] = 1;
while (i > 0) {
    A[i-1] = A[i]/2;
    i--;
}
ASSERT(∑_{k=0}^{n} A[k] = 2 − 1/2^n)
```

   (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.

ANSWER: I am going to make this a two-step process rather than just presenting the end tableau in the hope that this will give you a glimpse into the thought process that went into the said tableau. All the tableaux below are constructed bottom-up as usual.

First we start using the techniques we all know and love, strengthening the post-condition with the negated loop condition as well as the range for the loop variable `i`. We can go almost but not quite all the way to the top:

```
ASSERT(0 <= n <= max)
int i;
i = n;
A[i] = 1;
while (i > 0) {
    // aaand here is where we get stuck
    ASSERT(A[i]/2 + ∑_{k=i}^{n} A[k] == 2 − 1/2^{n−i+1} && 0 <= i − 1 <= n <= max)
    // remove the array component notation
    ASSERT((A|i − 1 ↦ A[i]/2)[i − 1] + ∑_{k=i}^{n} (A|i − 1 ↦ A[i]/2)[k] == 2 − 1/2^{n−i+1}
            && 0 <= i − 1 <= n <= max)
    // split the sum to eliminate the array component notation
    ASSERT(∑_{k=i−1}^{n} (A|i − 1 ↦ A[i]/2)[k] == 2 − 1/2^{n−i+1} && 0 <= i − 1 <= n <= max)
    A[i-1] = A[i]/2;
    ASSERT(∑_{k=i−1}^{n} A[k] == 2 − 1/2^{n−i+1} && 0 <= i − 1 <= n <= max)
    i--;
```

```
    ASSERT(∑_{k=i}^{n} A[k] == 2 − 1/2^{n−i} && 0 <= i <= n <= max)
}
ASSERT(∑_{k=i}^{n} A[k] == 2 − 1/2^{n−i} && 0 <= i <= n <= max && i <= 0)
// we know that i == 0 and that we need to introduce it on both sides of the ==
ASSERT(∑_{k=0}^{n} A[k] == 2 − 1/2^{n} && 0 <= i <= n <= max && i <= 0)
ASSERT(∑_{k=0}^{n} A[k] == 2 − 1/2^{n} && 0 <= i <= n <= max)
ASSERT(∑_{k=0}^{n} A[k] == 2 − 1/2^{n})
```

The key to getting unstuck is to figure out *why* we got stuck. I am not going to bother with the mismatch in the range, that is easily fixed using the standard techniques we have been using for awhile. The problem is that the assertion at the top of the loop include $\mathtt{A[i]}/2 + \sum_{\mathtt{k=i}}^{\mathtt{n}} \mathtt{A[k]} == 2 - 1/2^{\mathtt{n-i+1}}$, and that is definitely not the same as the corresponding bottom assertion namely, $\sum_{k=i}^{n} A[k] == 2 - 1/2^{n-i}$. Clearly, we need some creative strengthening.

We can further narrow the problem by looking at the top assertion and noticing that we have no idea what is the value of A[i]. Sure we know that A[i] contributes to the sum, but while we do know the sum, we have no idea what is the actual value of the individual contributors. We know however what we need to have at the top the same assertion ($\sum_{k=i}^{n} A[k] == 2 - 1/2^{n-i}$ plus any further terms we will strengthen with) that we started with at the bottom, so let us see what the value of A[i] *needs* to be for this to happen: If $\sum_{k=i}^{n} A[k] == 2 - 1/2^{n-i}$ then $\mathtt{A[i]}/2 + \sum_{\mathtt{k=i}}^{\mathtt{n}} \mathtt{A[k]} == 2 - 1/2^{\mathtt{n-i+1}}$ is equivalent to $\mathtt{A[i]}/2 + 2 - 1/2^{\mathtt{n-i}} == 2 - 1/2^{\mathtt{n-i+1}}$, which means $\mathtt{A[i]}/2 == 1/2^{\mathtt{n-i}} - 1/2^{\mathtt{n-i+1}}$, or $\mathtt{A[i]}/2 == 1/2^{\mathtt{n-i+1}}$ that is, $\mathtt{A[i]} == 1/2^{\mathtt{n-i}}$. If we can establish this then all our troubles are over.

But wait, we can establish as many new assertions as we wish by just stating them as new conjunctive factors. All we need to do is go back to the loop post-condition, strengthen that by adding $\mathtt{A[i]} == 1/2^{\mathtt{n-i}}$, which will be carried over throughout the proof all the way to the top. The only thing that needs to be kept in mind is that once you add a new conjunctive term you are stuck with it and cannot remove later. In other words, such an added term must be carried over all the way to the very top. Fortunately this is not going to be a problem for us since the extra statement will conveniently disappear in the end (like all the nice, true assertions do). Here is the complete tableau:

```
ASSERT(0 <= n <= max)
int i;
ASSERT(0 <= n <= max)
ASSERT(1 == 1/2^0 && 1 == 2 − 1/2^0 && 0 <= n <= max)
ASSERT(1 == 1/2^0 && (A|n ↦ 1)[n] == 2 − 1/2^{n−n} && 0 <= n <= n <= max)
ASSERT(1 == 1/2^{n−n} && ∑_{k=n}^{n}(A|n ↦ 1)[k] == 2 − 1/2^{n−n} && 0 <= n <= n <= max)
i = n;
ASSERT(1 == 1/2^{n−i} && ∑_{k=i}^{n}(A|i ↦ 1)[k] == 2 − 1/2^{n−i} && 0 <= i <= n <= max)
ASSERT((A|i ↦ 1)[i] == 1/2^{n−i} && ∑_{k=i}^{n}(A|i ↦ 1)[k] == 2 − 1/2^{n−i} && 0 <= i <= n <= max)
A[i] = 1;
ASSERT(A[i] == 1/2^{n−i} && ∑_{k=i}^{n} A[k] == 2 − 1/2^{n−i} && 0 <= i <= n <= max)
while (i > 0) {
    ASSERT(A[i] == 1/2^{n−i} && ∑_{k=i}^{n} A[k] == 2 − 1/2^{n−i} && 0 <= i <= n <= max && i > 0)
```

```
    FACT(0 < i && 0 <= i    =>    1 <= i    <=>    0 <= i − 1)
    FACT(i <= n    =>    i − 1 <= n)
    ASSERT(A[i] == 1/2^(n−i) && ∑_{k=i}^{n} A[k] == 2 − 1/2^(n−i) && 0 <= i − 1 <= n <= max && i > 0)
    ASSERT(A[i] == 1/2^(n−i) && ∑_{k=i}^{n} A[k] == 2 − 1/2^(n−i) && 0 <= i − 1 <= n <= max)
    ASSERT(A[i] == 1/2^(n−i) && ∑_{k=i}^{n} A[k] == 2 − 1/2^(n−i+1) − 1/2^(n−i+1) && 0 <= i − 1 <= n <= max)
    ASSERT(A[i] == 1/2^(n−i) && 1/2^(n−i)/2 + ∑_{k=i}^{n} A[k] == 2 − 1/2^(n−i+1) && 0 <= i − 1 <= n <= max)
    ASSERT(A[i] == 1/2^(n−i) && A[i]/2 + ∑_{k=i}^{n} A[k] == 2 − 1/2^(n−i+1) && 0 <= i − 1 <= n <= max)
    ASSERT(A[i] == 1/2^(n−i) && (A|i − 1 ↦ A[i]/2)[i − 1] + ∑_{k=i}^{n}(A|i − 1 ↦ A[i]/2)[k] == 2 − 1/2^(n−i+1)
            && 0 <= i − 1 <= n <= max)
    ASSERT(A[i] == 1/2^(n−i) && ∑_{k=i−1}^{n}(A|i − 1 ↦ A[i]/2)[k] == 2 − 1/2^(n−i+1)
            && 0 <= i − 1 <= n <= max)
    ASSERT(A[i]/2 == 1/2^(n−i+1) && ∑_{k=i−1}^{n}(A|i − 1 ↦ A[i]/2)[k] == 2 − 1/2^(n−i+1)
            && 0 <= i − 1 <= n <= max)
    ASSERT((A|i − 1 ↦ A[i]/2)[i − 1] == 1/2^(n−i+1) && ∑_{k=i−1}^{n}(A|i − 1 ↦ A[i]/2)[k] == 2 − 1/2^(n−i+1)
            && 0 <= i − 1 <= n <= max)
    A[i-1] = A[i]/2;
    ASSERT(A[i − 1] == 1/2^(n−i+1) && ∑_{k=i−1}^{n} A[k] == 2 − 1/2^(n−i+1) && 0 <= i − 1 <= n <= max)
    i--;
    ASSERT(A[i] == 1/2^(n−i) && ∑_{k=i}^{n} A[k] == 2 − 1/2^(n−i) && 0 <= i <= n <= max)
}
ASSERT(A[i] == 1/2^(n−i) && ∑_{k=i}^{n} A[k] == 2 − 1/2^(n−i) && 0 <= i <= n <= max && i <= 0)
// we know that i == 0
ASSERT(A[0] == 1/2^n && ∑_{k=0}^{n} A[k] == 2 − 1/2^n && 0 <= i <= n <= max && i <= 0)
// because I can, and it is also true, and I am also going to need it later
ASSERT(∑_{k=0}^{n} A[k] == 2 − 1/2^n && 0 <= i <= n <= max && i <= 0)
ASSERT(∑_{k=0}^{n} A[k] == 2 − 1/2^n && 0 <= i <= n <= max)
ASSERT(∑_{k=0}^{n} A[k] == 2 − 1/2^n)
```

I could not resist and went a bit fancier than what I said before by first strengthening with $A[0] == 1/2^n$ and then modifying that to $A[i] == 1/2^{n-i}$ when $i == 0$ is inferred. It is however perfectly fine to just introduce out of the blue $A[i] == 1/2^{n-i}$ in the loop post-condition, I just thought that the proof is cleaner this way (though obviously cleanliness is in the eye of the beholder).

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

ANSWER: The tableau from the previous question supports `i` as a loop variant. Indeed, before the loop `i` is assigned `n`, which is a positive value since `n>=0` according to the pre-condition. Secondly, `i` is monotonously and continuously decremented with every loop iteration, since the statement `i--;` is executed exactly once on each such an iteration. Finally, `i==0` implies that the loop condition `i>0` is false, and so the loop terminates. Having found the variant `i` we conclude that the loop terminates and so the given correctness statement is totally correct.

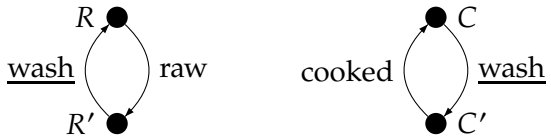
---

2. 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.

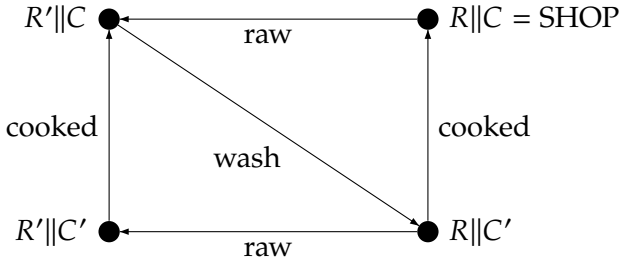
$$\begin{array}{lcl} \text{RAW} & = & \text{raw} \rightarrow \text{wash} \rightarrow \text{RAW} \\ \text{COOKED} & = & \text{wash} \rightarrow \text{cooked} \rightarrow \text{COOKED} \\ \text{SHOP} & = & \text{RAW} \parallel \text{COOKED} \end{array}$$

(a) Draw the transition graph of the process SHOP.

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



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



(b) Give the set traces(SHOP) and explain how you computed it.

Answer: For brevity and clarity we shorten raw, cooked, and wash to the single symbols r, c, and w, respectively. This way we can also use normal regular expressions for the set traces(SHOP).

We identify the main cycle first: we can go from SHOP back into SHOP doing rwc. Additionally, after performing rw we can take the side trip rcw before continuing, and we can do so repeatedly. There is no other way to come back, so all the possible round trips to/from SHOP are given by the regular expression $rw(rcw)^*c$. We go through the cycle as many times as we wish and then we finish by performing a last leg that generates some prefix of $rw(rcw)^*c$. The possible such prefixes are ε, r, rw, rwr, $rwrc$, and $rw(rcw)^*$, and we have to include them at the end of our traces. Overall we have:

$$\text{traces(SHOP)} = (rw(rcw)^*c)^*(\varepsilon + r + rw + rwr + rwrc + rw(rcw)^*)$$

(c) 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 by analyzing the traces of SHOP.

ANSWER: We note the fact that rc (handling two kinds of meat without a hand wash in between) appears explicitly in the loop body $rw(rcw)^*c$. The sequence cr can also be generated if we loop two times (when the trailing c of the first iteration comes immediately before the leading r of the second iteration). This all means that it is possible to handle raw meat before cooked meat (and also the other way around) without any intervening hand washing. Therefore the shop does not meet the given hygiene requirement.

Make sure you review the submission guidelines posted on the course's Web site before handing in your answers.