Merge partial correctness

supplement to W7-8 lecture

This document contains a tidied up version of a partial correctness proof that we covered in lecture (weeks 7 & 8). Recall the following function merge, which is a helper function used by mergesort:

```
def merge(A, B):
  """Pre: A and B are sorted lists of numbers.
  Post: return a sorted permutation of A+B
  """
  i = j = 0
  C = []
  while i < len(A) and j < len(B):
    if A[i] <= B[j]:
      C.append(A[i])
      i += 1
    else:
      C.append(B[j])
      j += 1
  return C + A[i:] + B[j:]
```

We prove the partial correctness of merge in two parts. Theorem 1 proves some loop invariants (facts which are true at the end of every iteration of the loop - including the 'zeroth' iteration, which is the state of the program before entering the loop). Theorem 2 uses those loop invariants to prove that, if merge terminates, then it satisfies the postcondition.

In lecture, we tackled this problem in the following order:

1. Brainstormed loop invariants
2. Wrote the partial correctness proof (assuming the invariants we came up with in the previous step were true)
3. Proved the loop invariants, by induction

We chose this order for a good reason. When we wrote our partial correctness proof, we only ended up using about half of the invariants we had cooked up in the first step. Because invariants are just a means to an end (partial correctness, in this case), we were able to cross off the ones we didn't need, and only prove the ones we needed. These are the three that are presented in Theorem 1 below.

Theorem 1 (Loop invariants). *At the end of each iteration k*

(a) C_k *is sorted*

(b) C_k *is a permutation of* $A[: i_k] + B[: j_k]$

(c) Every element in C_k is $\leq$ every element in $A[i_k :]$ and $B[j_k :]$

Proof. $C_0 = [\,]$, so (a) and (c) are trivially true before the first iteration. (b) is also true because $A[: i_0] + B[: j_0] = A[: 0] + B[: 0] = [\,]$.

Assume our invariant holds at the end of some iteration k, and that a $k+1$th iteration occurs. There are two possibilities depending on the if condition on line 8:

Case 1:

- $A[i_k] \leq B[j_k]$
- $C_{k+1} = C_k + [A[i_k]]$
- $i_{k+1} = i_k + 1$
- $j_{k+1} = j_k$

Case 2:

- $A[i_k] > B[j_k]$
- $C_{k+1} = C_k + [B[j_k]]$
- $i_{k+1} = i_k$
- $j_{k+1} = j_k + 1$

I will prove that in both cases, all three invariants are preserved.

By the IH, C_k is sorted. After appending either $A[i_k]$ or $B[j_k]$, C will still be sorted, because of our assumption that (c) holds at the end of the kth iteration. So (a) holds.

In case 1, $A[: i_{k+1}] + B[: j_{k+1}] = A[: i_k] + [A[i_k]] + B[: j_k]$. In other words, the list that we want C_{k+1} to be a permutation of differs from the version from the previous iteration by the addition of one element: $A[i_k]$. This is precisely the element we append to C_k to form C_{k+1}. A parallel argument applies to case 2. So (b) holds.

For (c) to be preserved, we must show that the newly appended element is $\leq$ all elements in the slices $A[i_{k+1} :]$ and $B[j_{k+1} :]$. If the newly appended element is $A[i_k]$ (case 1), then it is less than every element in $A[i_{k+1} :] = A[i_k + 1 :]$, since A is sorted (by the precondition). It is also no greater than $B[j_{k+1}]$ by the outcome of the 'if'. Because B is also sorted, this means that $A[i_k]$ is no greater than $B[j_{k+1} :]$. A parallel argument applies to case 2 (the only difference is that the 'if' outcome leads to a strict inequality, but this doesn't alter our argument). So (c) holds. □

Theorem 2 (Partial correctness). *If merge terminates, then it satisfies the postcondition.*

Proof. Suppose the while loop exits after some number of iterations k.

By invariant (b) C_k is a permutation of $A[: i_k] + B[: j_k]$. Since $A[i_k :] + B[j_k :]$ comprise the 'leftover' elements from A and B not contained in C_k, it follows that our return value is a permutation of $A + B$. It remains to show that it is sorted.

By the loop condition (line 7), $i_k \geq \text{len}(A) \vee j_k \geq \text{len}(B)$, so at least one of the slices concatenated on line 14 is empty. Without loss of generality, assume $B[j_k :]$ is empty.[1] Thus we return $C_k + A[i_k :]$.

By invariant (a), C_k is sorted. By the precondition, $A[i_k :]$ is sorted. Finally, by invariant (c), every element of C_k is $\leq$ every element in $A[i_k :]$. These facts together imply that the return value is sorted. □

[1] By saying this, we're claiming that the rest of the proof is not using any special properties that differentiate B from A. i.e. we could do a find-and-replace to swap all references to B and A and our logic would still hold. We used a similar trick twice in our proof of the loop invariants above when we said that "a parallel argument applies to case 2".