CSE 331: Software Design & Implementation

Section 7

The following functions, "with" and "without", each take two arrays, L and R, as input. These arrays are required to be **sorted** and have **distinct** elements. In other words, we must have

$$L[0] < L[1] < \cdots < L[n-1] \quad \text{and} \quad R[0] < R[1] < \cdots < R[m-1]$$

where n is the length of L and m is the length of R.

The call "with(L, R)" returns the elements that appear in either L or R (or both), combined in sorted order. That function is defined as follows:

$$\begin{array}{lll}
\textbf{func}\ \text{with}([], []) & := & [] \\
\quad \text{with}([], R + [y]) & := & \text{with}([], R) + [y] \\
\quad \text{with}(L + [x], []) & := & \text{with}(L, []) + [x] \\
\quad \text{with}(L + [x], R + [y]) & := & \text{with}(L + [x], R) + [y] \quad \text{if } x < y \\
\quad \text{with}(L + [x], R + [y]) & := & \text{with}(L, R) + [x] \quad \text{if } x = y \\
\quad \text{with}(L + [x], R + [y]) & := & \text{with}(L, R + [y]) + [x] \quad \text{if } x > y
\end{array}$$

where each of these rules is for any $x, y : \mathbb{Z}$ and any $L, R : \text{Array}_{\mathbb{Z}}$.

The call "without(L, R)" returns the elements of L that *do not* appear in R, in the same order as they came in L. That function is defined as follows:

$$\begin{array}{lll}
\textbf{func}\ \text{without}([], R) & := & [] \\
\quad \text{without}(L + [x], []) & := & \text{without}(L, []) + [x] \\
\quad \text{without}(L + [x], R + [y]) & := & \text{without}(L + [x], R) \quad \text{if } x < y \\
\quad \text{without}(L + [x], R + [y]) & := & \text{without}(L, R) \quad \text{if } x = y \\
\quad \text{without}(L + [x], R + [y]) & := & \text{without}(L, R + [y]) + [x] \quad \text{if } x > y
\end{array}$$

where each of these rules is for any $x, y : \mathbb{Z}$ and any $L, R : \text{Array}_{\mathbb{Z}}$.

1. It's Without Time

(a) Prove by induction that without$(L, []) = L$ for any array L.

(b) Is it true that with$(L, []) = L$ for any array L? Why or why not?

(c) Is it true that with$([], R) = R$ for any array R? Why or why not?

2. Keep a Sharp Without

(a) Prove by calculation that without$([1,2,3],[2,4]) = [1,3]$.

(b) The definition of "without" drops the last element $[y]$ from R on the recursive call when $x < y$ or $x = y$ but not when $x > y$. Suppose that we changed the rule when $x > y$ to the following:

$$\text{without}(L +\!\!+ [x], R +\!\!+ [y]) \quad := \quad \text{without}(L, R) +\!\!+ [x]$$

so that $[y]$ is dropped in this case as well.

Show that we would then get the wrong answer for without$([1,2,3],[2,4])$.

(c) The recursive calls drop elements from the ends of the two arrays, so each recursive call, besides the last, is of the form "without$(L[0\,..\,i], R[0\,..\,j])$" for some integers $i, j \geq 0$. This is true both with the original definition and with the changed (incorrect) definition in part (b). There was something about the first definition that made it work, while the second definition did not.

What condition about how $L[i]$ relates to R on these recursive calls does the original definition guarantee that the changed definition does not?

Hint: have a look at where the calculation in (b) went off the rails.

One advantage of loops over invariants is that this condition from part (c) actually has a place to live: it should be part of the loop invariant. In contrast, this condition sort of "disappears into the ether" in the recursive version. It is important for understanding why the recursive version works, but it is not stated anywhere!

3. By Points or By Without

Prove by induction on S that, if $L[i]$ is less than every element of S (i.e., $L[i] < S[k]$ for $k = 0 \mathinner{..} S.\text{length} - 1$) and both L and $R + S$ are sorted and contain distinct elements, then we have

$$\text{without}(L[0 \mathinner{..} i], R + S) = \text{without}(L[0 \mathinner{..} i], R)$$

In the previous problem, we saw that the elements greater than $L[i]$ are the only ones that it is safe to drop. In this problem, we are proving that it is okay to drop all of those elements.

Note that the set of all arrays S satisfying the conditions above can be defined recursively as follows. "$[]$" is in S, and if a list T is in S, then $T + [x]$ is in S for any integer x that is larger than $L[i]$ and larger than every element of T. (This second part ensures that $T + [x]$ is sorted and contains distinct elements. The first part ensures that the condition "every element of $T + [x]$ is larger than $L[i]$" holds. Furthermore, any array S satisfying these conditions can be built up in this way.)

4. Set Off Some Steam

One use of sorted arrays of distinct elements is to represent sets of numbers. Sets only care about whether an element is present or not, so when storing the numbers in an array, we are free to choose any order we want, and as we just saw, sorted order offers important benefits for calculating operations like "with" and "without".

These two operations have immediate applications to sets. If A and B are sets represented as sorted arrays, then with(A, B) is the set of numbers in either A or B (or both), which we call the "union". The operation without(A, B) is called the "difference" of the sets A and B.

(a) Given our set representation with a sorted list, what is an algorithm that we could use to determine whether a number exists in our set or not? (i.e. what can we use to implement the contains() function of our set)

(b) The "intersection" of sets A and B contains the numbers that appear in both sets. How would we calculate the intersection if A and B are sorted arrays? (Hint: you'll need more than one function call!)

(c) The "complement" of a set A with respect to a set U (which must contain all the elements of A) is the set of numbers in U but not in A. How would we calculate the complement of A with respect to U if both A and U are sorted arrays?

(d) Using your answers to (a-b), how could you calculate the union of A with the complement of B with respect to U (that contains both A and B)? Is there another way that we could write this as the complement of some (other) set?