CS 199-173

Final Homework

December 9, 2021

Due: December 17

In this exercise, we will write a recursive function in Haskell and then use proof by induction to argue that it works correctly.

0. Watch this video about working with lists in Haskell (https://mediaspace.illinois.edu/media/t/1_sq9blnz7). Get the starter code from the Piazza post and confirm that it runs (as with HW 2, you can use an online interpreter or you can install Haskell locally).

1. Write a recursive function "lcsLen" which takes two lists, and returns the *length* of some *longest common sublist*. (A *sublist* is a sequence of values that appears in order in a list, not necessarily consecutively, e.g. [3,5] is a sublist of [6,1,3,7,5]. A *common sublist* for two lists is a list that is a sublist of each of them, e.g. [3,5] is a common sublist of [6,1,3,7,5] and [1,9,6,2,3,8,4,5]. Note that *longest common sublists* are not necessarily unique, e.g. both [6,3,5] and [1,3,5] are longest common sublists of the two lists above. But by definition, for any pair of input lists, all their longest common sublists will have the same length.) Do not use any Haskell functions other than comparisons, arithmetic, cons (:), and max.

   (Hint: You may want multiple base cases.)

   (Hint: You may want separate recursive cases for when the first items in the two lists are the same and when they are different. You may need more than one recursive call in some cases.)

   (Note: you do not need to come up with an efficient *algorithm here. The naive algorithm is exponential time, but it can be made much faster using a technique called "dynamic programming", where we avoid recomputing things we've done before by recording the results of various calls. You will study this technique in CS374.)*

2. Write a proof by induction that your "lcsLen" algorithm is correct, i.e. a proof of the the following claim: "For any lists l_1 and l_2, lcsLen returns the length of some longest common sublist." To make the proof easier you may assume that within a list there are no duplicate values (but your *code* should still work even if there are duplicates). *(Make sure your 'induction variable' is something that decreases in every recursive call.)*