Jalil Farid
Programming Assignment 3

A. To produce an algorithm to decide whether a string is an interleaving, one can choose an implementation using ideas from non-deterministic finite automata seen in Ullman et. al.'s Automata Theory. Specifically, let x and y be the base string for the repeated signals, and assume we have a test string t which may or may not be an interleaving. We start by building a deterministic finite automata to accept repetitions for each string, adding a state for each letter and a transition between adjacent letters, as well as a final transition from the last letter to the start letter. Further, we can combine each repetition FSA into a product FSA which keeps the states of each FSA, and each transition on a character must transition on either the FSA for string x or the FSA for string y. While this construction is simple, the problem is this: given a character which may transition on either the FSA for x or the FSA for y, explicitly choosing one path may lead to a later state which does not produce an untangling. When given the choice of a transition to a new state on both FSAs, we must choose both to try to find a decision as to whether or not the test string can be untangled. This leads to the construction of a non-deterministic finite automata, which can be in several states at one time. To extend the product FSA, we simply add the ability to be in multiple states at any given time. Formally, this construction will require an "end" state, but for the purposes of the problem, we do not require the FSAs to have ended on their base string patterns because the repetition string is required to be a prefix of concatenations of base strings.

To analyze this procedure we start with n being the length of the test string t, and we assume the base strings x and y have size $|x| < n$. To start, the construction of the FSA for the respective strings is $O(n)$, since we must store and produce a transition for each input character in the base string. The run time complexity of testing can be tricky. For starters, the space complexity of storing the states is $O(n^2)$ since by definition for NDFA, there may be any number of current states less than or equal to the total number of available states, and we note that because of the product of each FSA of size $O(n)$, we have a product FSA with $O(n^2)$ states, giving the space complexity of the NDFA as $O(n^2)$. During the running of the NDFA, since there are $O(n^2)$ states, and $O(n)$ input characters in the test string, each character may need to transition on up to $O(n^2)$ states, and for each state, there may be up to 2 new states created, leaving a total run time of each transition as $O(2^{O(n^3)}) \approx EXPTIME$. While this is a worst case analysis, note that the NDFA will transition to "dead" states which cannot continue from either FSA state when given an input character that cannot transition. In practice, the running time may be much less, but we cannot know a priori how many transitions will lead to dead states.

B. The included code shows two test cases. One is a derived test case, and the other is the given test case in the assignment. The code specifically returns a tuple that counts the number of transitions, as well as a string representing the assignment of assignments of state transitions, a boolean representing whether ot not a interleaving was found, and an interleaving represented by

Jalil Farid
Programming Assignment 3

a list of “x”/”y” pairs encoding which repetition the character is assigned to, much like the given assignment. One can run the code for the given test case alone and see there are multiple solutions for the interleaving. Along with this, a plot is generated, which is included below, showing the scale of the number of transitions. Note for the test case 1, n=70 produces about 200M transitions. While this is much less than 2^{70^3}, we can see the scale for which the number of transitions rapidly rises. Note again, we don’t actually transition on all of the possible states because of dead states.