Assignment 1, Advanced Functional Programming

Casper Strömberg

November 2020

1 Graph Coloring

1.1 Data Structure

The graph data structure consists of a list with tuples where each tuple includes a label which represents a node and list of adjacent labels. The data structure is chosen due to its simplicity to implement.

1.2 Properties Tested

We start of by generating arbitrary lists that are connected in arbitrary ways. To create the graph we first create a random set of integers that will act as the labels in the graph. Then we generate all the possible edges that the graph could have, we take a random number of edges. Now we have a list of labels and a list of edges that we can create a graph from. We take these two lists and combine them in a way so that the vertexes that the edges depend on is included before the vertexes. Then we can just fold over that combined list and if we encounter a vertex we add a vertex and if we encounter an edge we add that to the graph.

To test the graph implementation we can compare the items that we added when generating the graph and the actual graph. We do that for the vertices and edges.

1.3 Lazy Monadic

We create all possible color combinations by running replicateM numNodes [1 .. numColors]. This will create a list of lists with possible solutions. We then use find to find the first solution that is safe ie. no neighbors has the same color. The laziness means that we can end the evaluation as soon as the we find a valid solution. The find function returns a maybe data type. We use the monadic properties to program for the best case. That is, if we don't find any solution we return nothing immediately without running trough the chain of expressions below. Therefore the rest of the implementation does not have to deal with the value being a maybe value.

1.4 Testing K-Coloring

We use the same method of generating a graph when testing the algorithm. We test that the number of colors return are never larger that the max number of colors specified in the input. We also test that if we provide as many colors as we have vertices we should always get a valid solution. We test that all solutions that we get are valid, ie that neighbors don't share the same value.