Exercise 9

We consider now the problem of modelling with Haskell programs the movements of an acrobat who walks on a rope with a pole in his hands. Nasty birds land on either end of the pole while the acrobat is moving, and he can keep his balance as long as the difference between the number of birds on the two ends is not larger than 4. The exercise asks to model this problem in 3 different ways as described below. First we need to introduce some types.

type Conf = (Int,Int) – the 2 integers are the number of birds on the left and on the right side of the pole. Such numbers should never be negative.

type Move = (Int, Char)—the Char can be either ‘L’ or ‘R’, for Left and Right, anche the integer is the number of birds landing (or flying away when it is negative) on (from) that side.

You are supposed to specify 3 functions play1,2, and 3 as follows:

1) play1 :: [Move] -> (Conf -> [Conf])
   thus play1, given a sequence of moves ms, should return a function that computes, for any start configuration s, the list of the configurations that are reached from s executing ms. play1 should be defined by using foldr. You may get inspiration for play1 from an example of a function executing moves on the plane and using foldr, cf. Lesson 6. Note that play1 does not express possible failure (the acrobat falling down) in its type. Thus it will use the builtin exception “error” to model failure.

2) play2:: [Move]-> (Conf -> Maybe [Conf]) the change with respect to play1 is that the function returned by play2, is such that for some start configuration may return Nothing, corresponding to sequence of moves that cause the acrobat to fall down. For such sequences of moves, play1 would terminate with an error message. Also play2 should be defined using foldr.

3) play3 :: Maybe Conf -> [Move] -> Maybe [Conf], thus play3 does not return a funzion (Conf -> Maybe [Conf]), but, it needs in input a particular start configuration s and a list of moves ms and it returns either Nothing, when ms, starting from s, leads to failure, or the list of configurations reached from s executing ms, when the acrobat keeps his balance. Play3 should be definide using the operations of Applicative, i.e. pure and <*>. This is possible because Maybe is Applicative.

   Notice that play1 and 2 are more higher-order that play3, in the sense that they return a function (Conf -> Maybe [Conf]) whereas play3 returns just the codomain of that function.

A file called Walk.hs must be turned in that contains the module Walk which comprises the 3 functions and all types and auxiliary functions thet they need. Alco the types Conf and Move, given above, must be included in this module.

In the moodle, for exercise 9 there is a main that calls the 3 functions and prints their result. Actually play1 is called last, because it fails in case of a sequence of moves that causes the acrobat to fall.