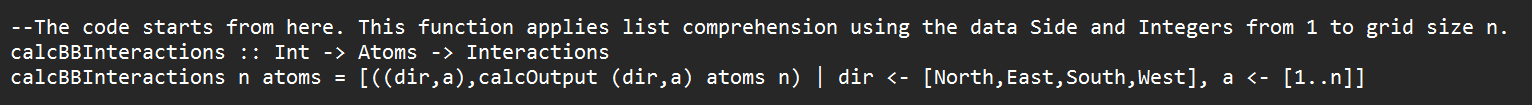
CalculateAllInteractions.hs

Explanation

This code uses the number of atoms to generate all the interactions from all entry points of the grid. It works in the way that the ray moves inside the grid one by one, depending on the condition, will be absorbed, reflected or reach a different end of the grid.

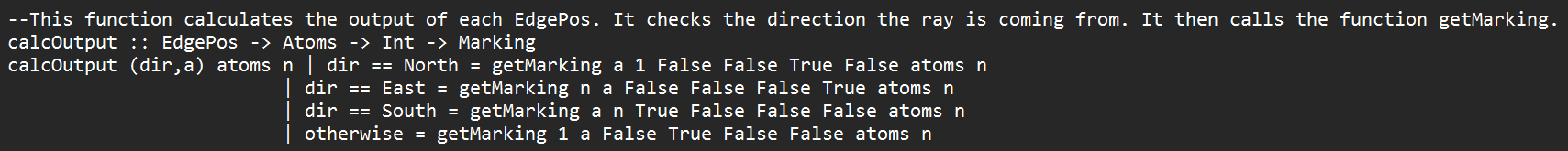
The code begins by calling the function ***calcBBInteractions***. This function takes in the inputs of the grid size “n” and the list of atoms “atoms” that are inside the grid. It applies list comprehension to generate interactions from all entry points in the grid.

In this function, “dir” represents the elements of data Side, and “a” represents the numbers from 1 to n. Both of these variables will be used to form a pair, and it will be printed on the left side of the interaction tuple, while it will be used as an input for the function “calcOutput” along with “atoms” and “n” on the right side of the interaction tuple. The function is shown below.



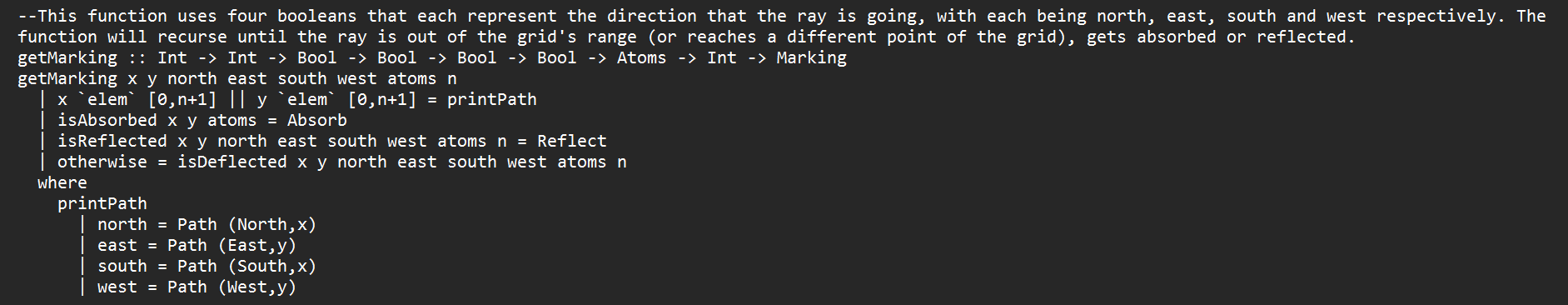
Moving on to the next function ***calcOutput***. This function takes in the inputs of the EdgePos pair “(dir,a)”, the list of atoms “atoms” and the grid size “n”. The function first checks which direction the ray is coming from. Depending on the direction, the function “getMarking” will be called. If the direction the ray coming from is North, then it will input that the ray is going south, if it is South, then it will input north, if it is West, then it will input east and if it is East, then it will input West. Below is the function.

In this function, guards are used to check the direction the ray is coming from. Then, depending on the starting direction, it will call the function “getMarking” with different input values. Below is the function.

 The last function of the code ***getMarking***, returns the output of the code. The inputs include:

* The first two integers represents the current position of the ray (colNum,rowNum).
  + North starts from (colNum,1)
  + South starts from (colNum,grid size)
  + East starts from (grid size,rowNum)
  + West starts from (1,rowNum)
* The four Booleans represent the direction the ray is going, with that being north, east, south and west respectively. In any occasion, only one can be true and the rest being false.
* The list of atoms.
* The grid size.

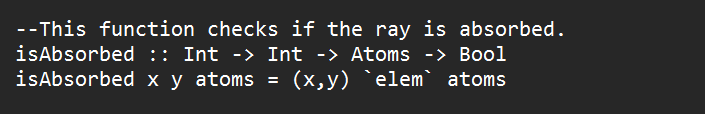
This function uses recursion to recursively calculate the path of the ray until it is absorbed, reflected or reaches an exit of the grid. If the nothing happens to the ray, the function will be recursed with the position of the ray being incremented depending on the direction of the ray heading. The function is shown below.



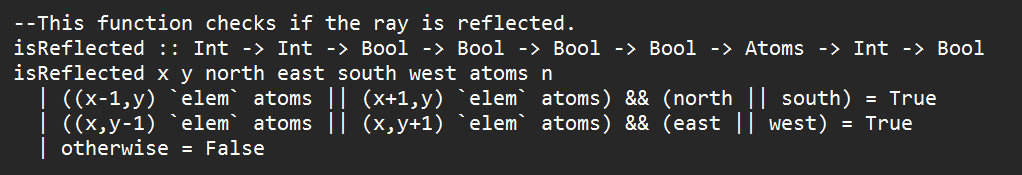
There are a total of 4 guards in this function. Each guard is respectively responsible for:

* checking whether the ray position is out of bound, if yes then the function will stop recursing and print the path the ray stops at.
* checking whether the ray gets absorbed by an atom, if yes then the function will return “Absorb”.
* checking whether the ray gets reflected by an atom, if yes then the function will return “Reflect”.
* checking whether the ray gets deflected by an atom or continues going straight.

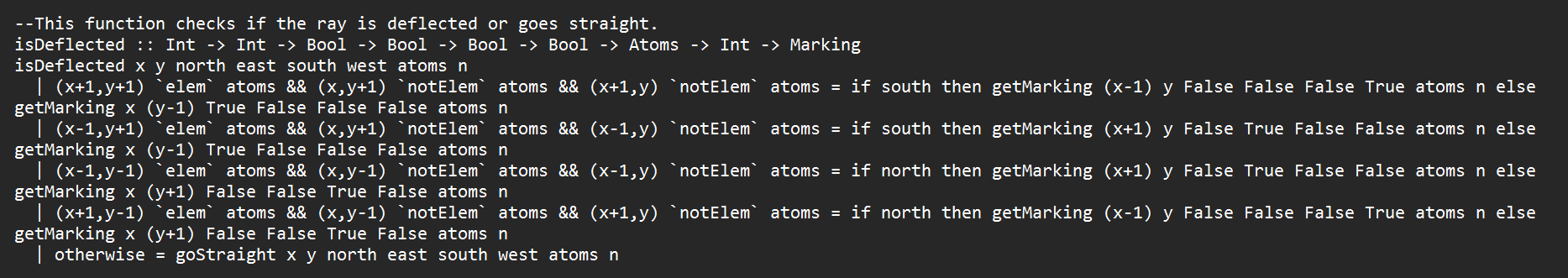
“isAbsorb”, “isReflected” and “isDeflected” are all separate functions outside “getMarking”.



***isAbsorb*** function checks if the ray position is equal to one of the atoms’ positions. If yes, then the function will return true and print Absorb, else it will continue with other functions.

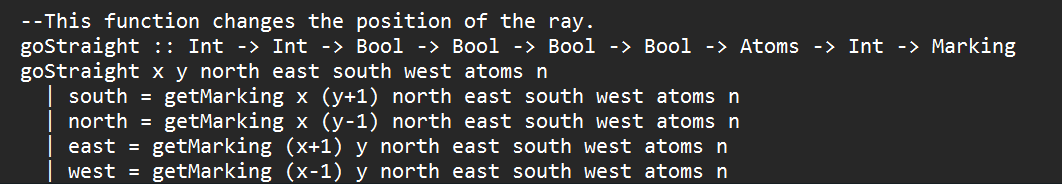


***isReflected*** function checks if the ray position is on the right or left of an atom, or on the top or bottom of an atom, if yes then the function will return true and print Reflect, else it will continue with other functions.



***isDeflected*** function has 5 guards, with each responsible for:

* checking whether there is an atom on the bottom right of the current ray position, and there is no atom on the right or bottom of the ray, if so then it checks the current direction of the ray, and if the ray direction is south, then the ray direction will become west, else if the ray direction is east, then the ray direction will become north.
* checking whether there is an atom on the bottom left of the current ray position, and there is no atom on the left or bottom of the ray, if so then it checks the current direction of the ray, and if the ray direction is south, then the ray direction will become east, else if the ray direction is west, then the ray direction will become north.
* checking whether there is an atom on the upper left of the current ray position, and there is no atom on the left or top of the ray, if so then it checks the current direction of the ray, and if the ray direction is north, then the ray direction will become east, else if the ray direction is west, then the ray direction will become south.
* checking whether there is an atom on the upper right of the current ray position, and there is no atom on the right or top of the ray, if so then it checks the current direction of the ray, and if the ray direction is north, then the ray direction will become west, else if the ray direction is east, then the ray direction will become south.
* checking if all the guards are not met, then the function “goStraight” will be called.

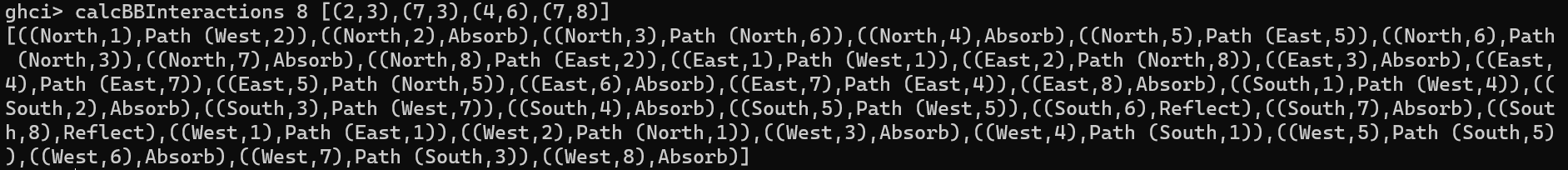


The function above ***goStraight*** is responsible for:

* checking if the current ray direction is south, then the function will call the function “getMarking” again with the rowNum increased by 1
* checking if the current ray direction is north, then the function will call the function “getMarking” again with the rowNum decreased by 1
* checking if the current ray direction is east, then the function will call the function “getMarking” again with the colNum increased by 1
* checking if the current ray direction is west, then the function will call the function “getMarking” again with the colNum decreased by 1

Output

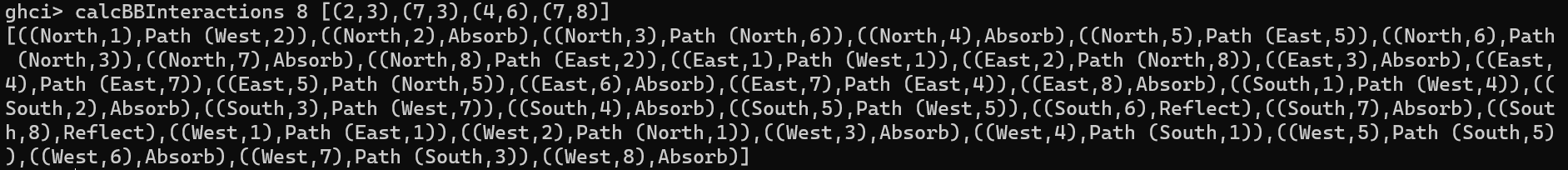
The code works by just running the function ***calcBBInteractions***. Only the grid size and the list of atoms will be inputted respectively, and a list of interactions will be generated. Here is how the input is entered and what the output generates.



For the function ***calcOutput***, it will take in inputs of EdgePos, which is in the form of a pair (Side,Int), and the list of atoms and the grid size. The function will print out the marking, which are either Absorb, Reflect or Path(EdgePos). Here is what happens if the following input is done.



When input (North,1) is entered, the output becomes Path(West,2). This is one of the answers when the function “calcBBInteractions” is ran.



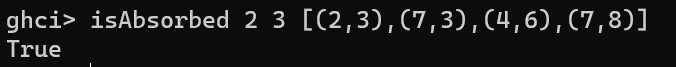


The function ***getMarking*** prints the output of the function “calcOutput”, or in other words, “calcOutput” uses “getMarking” to get the output. Instead, “getMarking” inputs two integers representing column number and row number respectively, four booleans representing north, east, south and west respectively, the list of atoms and the grid size. Here in the screenshot below, 1 1 means that the ray starts from (1,1) in the grid, and the output is the same as calcOutput (North,1) … .

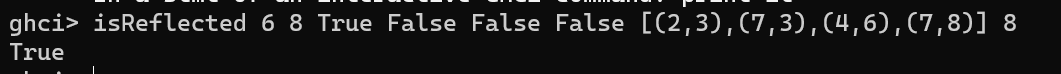


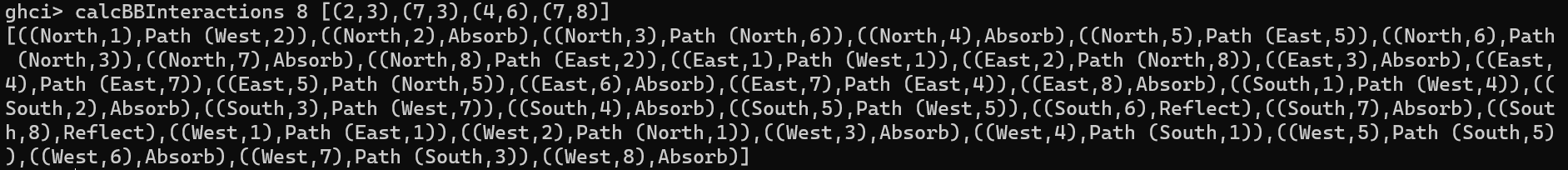
“GetMarking” contains three other functions inside, with them being ***isAbsorbed***, ***isReflected*** and ***isDeflected***. Each function is responsible for returning Absorb, Reflect, Path(Side,Int) or recurse again.

When both the column number and the row number are entered in the function “isAbsorbed”, then the function will check if the pair exists inside the atoms, if so, then it will return Absorb. In the screenshot below, 2 and 3 are the inputs, and the function checks if (2,3) is inside the atoms list.



When both the column number and the row number are entered in the function “isReflected”, then the function will check if the pair has an atom on the top, right, bottom or left, and it also checks the current direction the ray is going. In the screenshot below, 6 and 8 are the inputs, along with the ray going north, and because the ray is going north and it checks for (6-1,8) and (6+1,8) inside the atoms list, and because the pair exists inside the list, the function returns true. The input is the same as inputting (South,6) in the function “calcBBInteractions”.

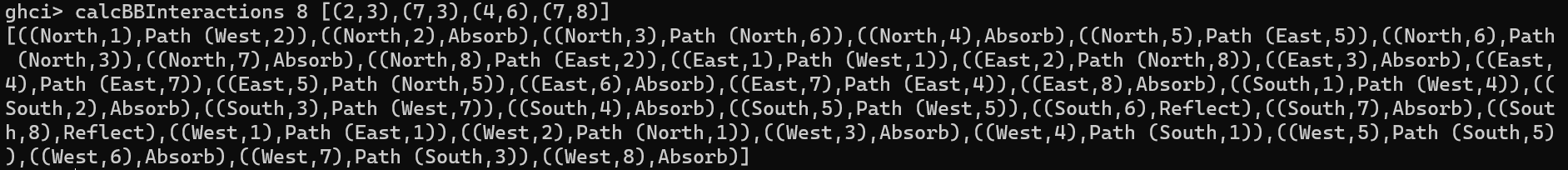






When both the column number and the row number are entered in the function “isDeflected”, then the function will check if the pair’s position is diagonal to an atom. If there is, then the function also checks the current ray direction, and it will deflect the ray perpendicularly. In the screenshot below, column number 5 and row number 1 are the inputs, along with the ray going south, and because the ray gets deflected by the atom at (4,6), the ray will travel east and end up at Path(East,5). The input is the same as inputting (North,5) in the function “calcBBInterations”.







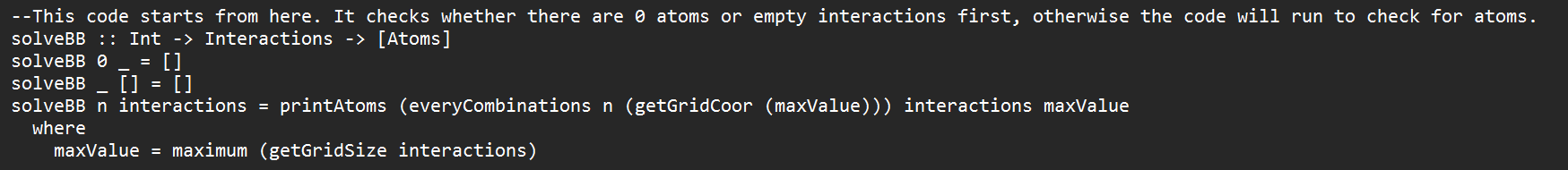
“isDeflected” function also uses the function ***goStraight*** for the conditions where the ray did not get deflected. The function also recurses back to the function “getMarking”.

SolveABlackBox.hs

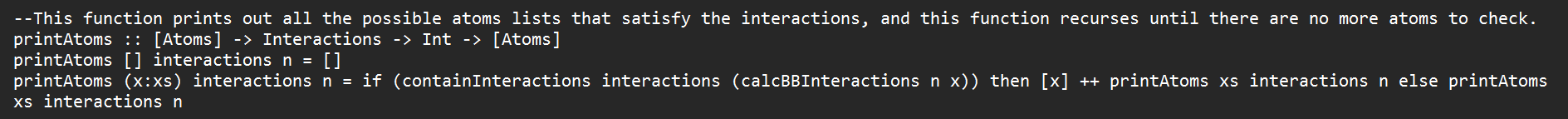
Explanation

This code uses the ray interactions to generated all the possible atoms. The code works by using brute-force method, to search for the list(s) of atoms that satisfy the interactions, and using the question “calcBBInteractions” from challenge to check for the list(s) of atoms.

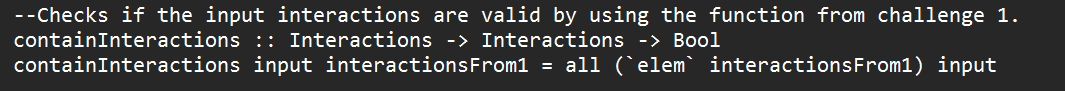
The code works by calling the function ***solveBB***. The function takes in the inputs of the number of atoms “n” and the list of interactions “interactions”. This function uses pattern matching to check whether “n” is 0 or “interactions” is empty, otherwise the function will return the desired output. Here is the function shown below.



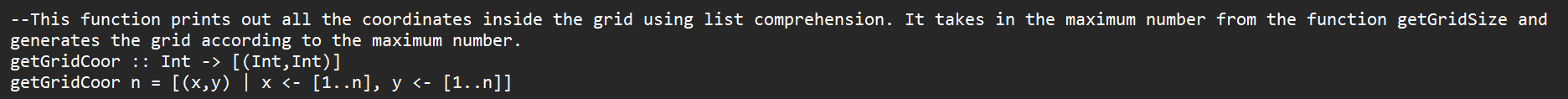
In this function, ***printAtoms*** is another function used to print out all the possible combinations of atoms, and the grid size “maxValue”, which is calculated from the interactions by getting the maximum value of the output of the function “getGridSize”, is one of the inputs. This function also inputs in all the list of atoms and interactions. This function uses recursion to print out the list of atoms that satisfy the interactions until the list “interactions” is empty. It also uses if then else statement for the function “containInteractions” to check if the answer is the same as the function “calcBBInteractions”. If it is the same, then the list of atoms will be printed and recursion will happen, otherwise the recursion will happen without the list of atoms being printed. The function is shown below.



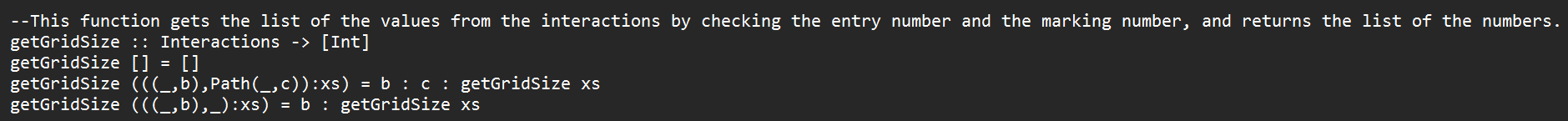
The function ***containInteractions*** takes in two inputs, the user input “interactions” and the output from the function “calcBBInteractions”. This function uses “all” to check if all the values from “interactions” are present in the output of “calcBBInteractions”. The function is shown below in the screenshot.



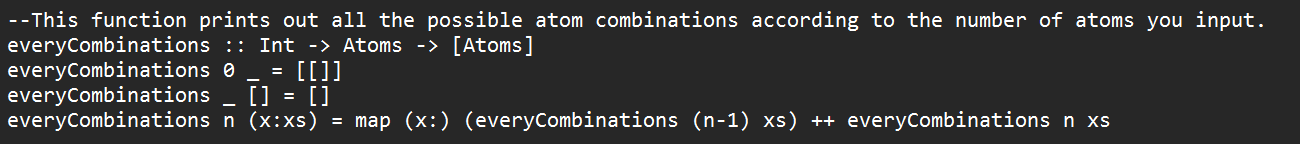
The next function ***getGridCoor*** generates all the coordinates inside the grid using list comprehension. The input is the maximum number from the function “getGridSize” and generates the grid according to the maximum number. Here is the function in the screenshot below.



The next function ***getGridSize*** gets the list of the values from the interactions by checking the entry number and the marking number, and returns the list of the numbers. It first checks if the interactions list is empty, otherwise it will use pattern matching to check for the entry and marking numbers. The numbers will then be appended to the final output and the function will recurse until there are no more values in the interactions list. The function is shown below.



The last function used in this code is ***everyCombinations***. This function prints out all the possible atom combinations according to the input as the number of atoms. The function checks if the integer input is zero or the atoms list “xs” is empty. If not, then the function will use map to print out all the possible atom combinations and recursion happens until the atoms list “xs” is empty. The screenshot below displays the function.

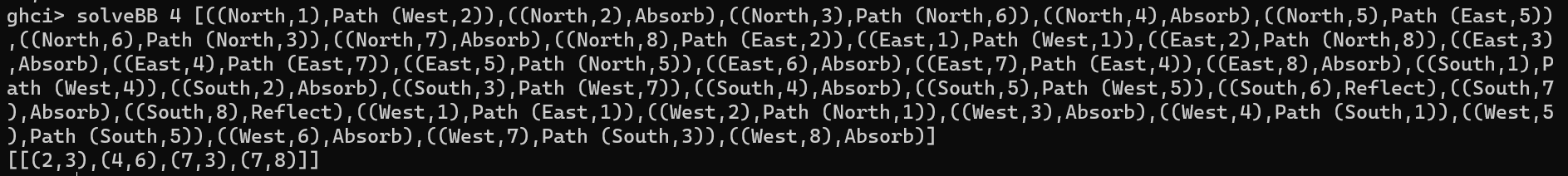


(Melpomene,2018)

The rest of the code is all from challenge 1.

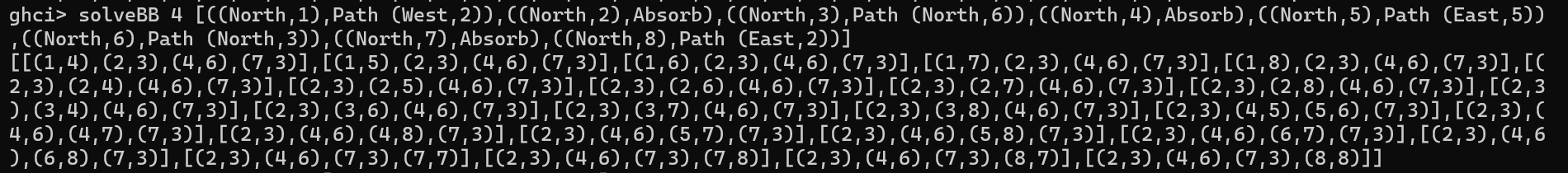
Output

The code works by just running the function ***solveBB***. Only the number of atoms and the list of interactions will be inputted respectively, and the list of atoms combinations will be generated. Here is when an example input is entered into the function.



Because this code uses brute-force method, the output will not be generated right away under some cases. The higher the number of atoms or the bigger the grid size in the interactions list entered, the longer time it will take for the output to be generated.

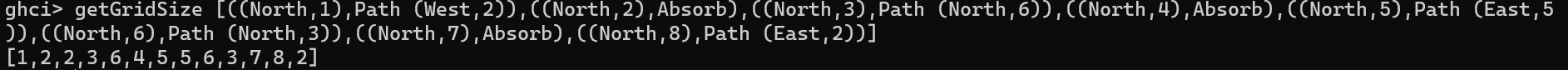
The function also generates atoms if the interactions are not incomplete. In the screenshot below, here is what happens if the interactions are not complete.



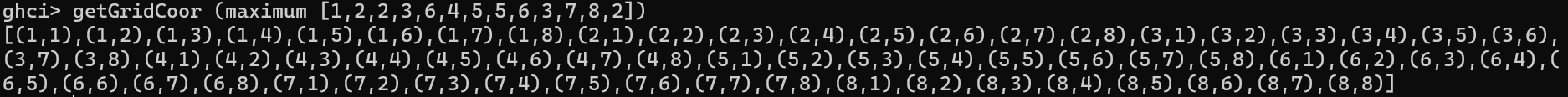
The function will print out the possible atoms combinations that satisfy the interactions.

The next function ***printAtoms*** is used for the function “solveBB”, so both of these function’s output will be the same. It takes in the inputs of the output from the function “everyCombinations”, the interactions list input by the user and the grid size.

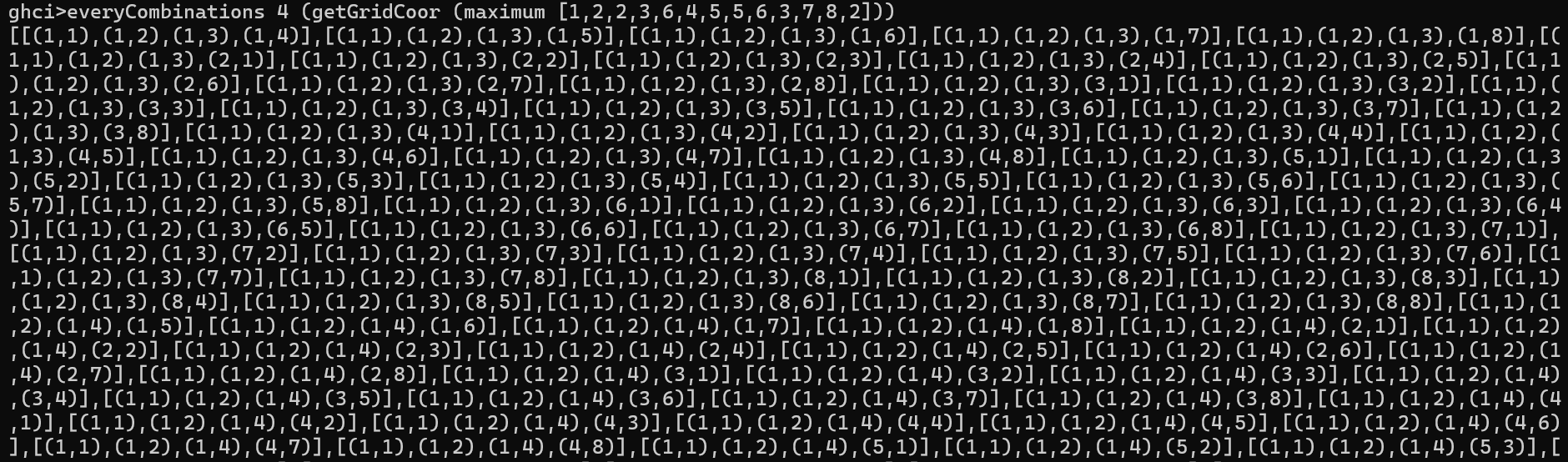
The function ***getgridSize*** uses the input “interactions” and gets the entry and marking number from every interaction value. In the screenshot below shows how the function works.



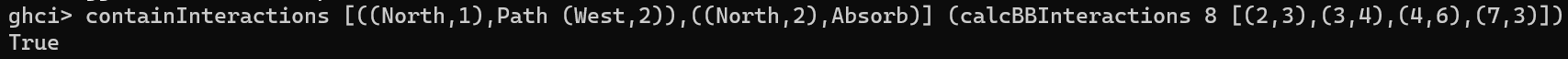
Afterwards, the maximum value will be checked and will be inputted into the function ***getGridCoor*** to print out the grid coordinate pairs inside the grid. It is shown below.



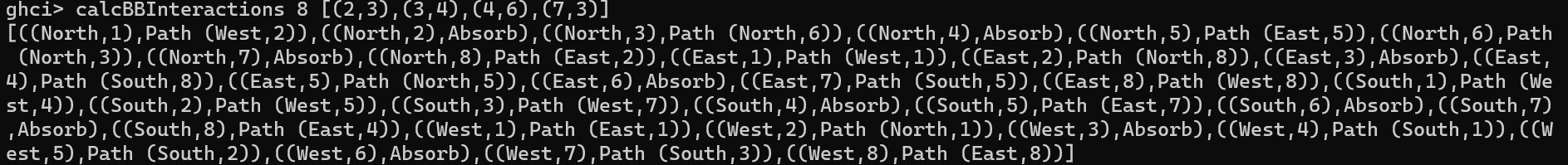
The function ***everyCombinations*** prints out all possible atom combinations depending on the inputs of the number of atoms and the output from “getGridCoor”. In the screenshot below shows the output. Because this function’s output is too long, the screenshot only shows the first few lines of the output, but the pattern remains.



Lastly, the function ***containInteractions*** gets the inputs of the interactions input from the user and the output of the function “calcBBInteractions”. An example input is entered and the output of the function is shown below.



Here is the output of “calcBBInteractions 8 [(2,3),(3,4),(4,6),(7,3)]”. It contains all the interaction values entered by the user.





Bibliography

Melpomene (2018), StackOverflow, *Available at:* [*https://stackoverflow.com/questions/52602474/function-to-generate-the-unique-combinations-of-a-list-in-haskell*](https://stackoverflow.com/questions/52602474/function-to-generate-the-unique-combinations-of-a-list-in-haskell) *(Accessed 5 Dec 2023)*