ASSIGNMENT 3 – FUNCTIONAL PROGRAMMING:

As my extension I wanted to create the algorithm which would take approximately the same amount of time to run as the initial Bombe algorithm but it would produce correct results much more often. I was kind of inspired by the first example from bombeTesting file:

```
p1 = "AIDEGHMC"
x1 = "TTCMAANO"
```

The initial algorithm quickly produced result to that crib: Just ((0,0,0),[('D','A'),('Z','C'),('K','O')])

When we check it using enigmaEncodeMessage, we get:

```
Index:                  0 1 2 3 4 5 6 7
Result :                M K C W F J K O
x1 :                    T T C M A A N O
Correct:                    +         +
```

As we can see, the produced steckerboard is obviously incorrect. That happens because we get the solution “if we reach the end of the menu without finding a contradiction“. That means that we check only the indexes in the longest menu in order to find those contradictions. In the example above the longest Menu is [2,7] so only letters at those positions are encoded correctly.

I had written a few different solutions to that problem before I found the best and the most efficient one.

At the beginning I’ve fixed that by using following approach:

We know that longest menu is [3,6,2,7,1]. Instead of checking only indexes in that list, 3 then 6 etc, we could:

1) Check for contradiction for a given index n

2) Check if letter in plain at index n occures somewhere in the plain. If not, move to the next index in the longest menu. If so, check for contradiction for the found indexes and then if no contradictions found, move to the next indext in the longest menu. E.g. in the following crib

```
p2="NONENEMC"
x2="TTCMAANO"
```

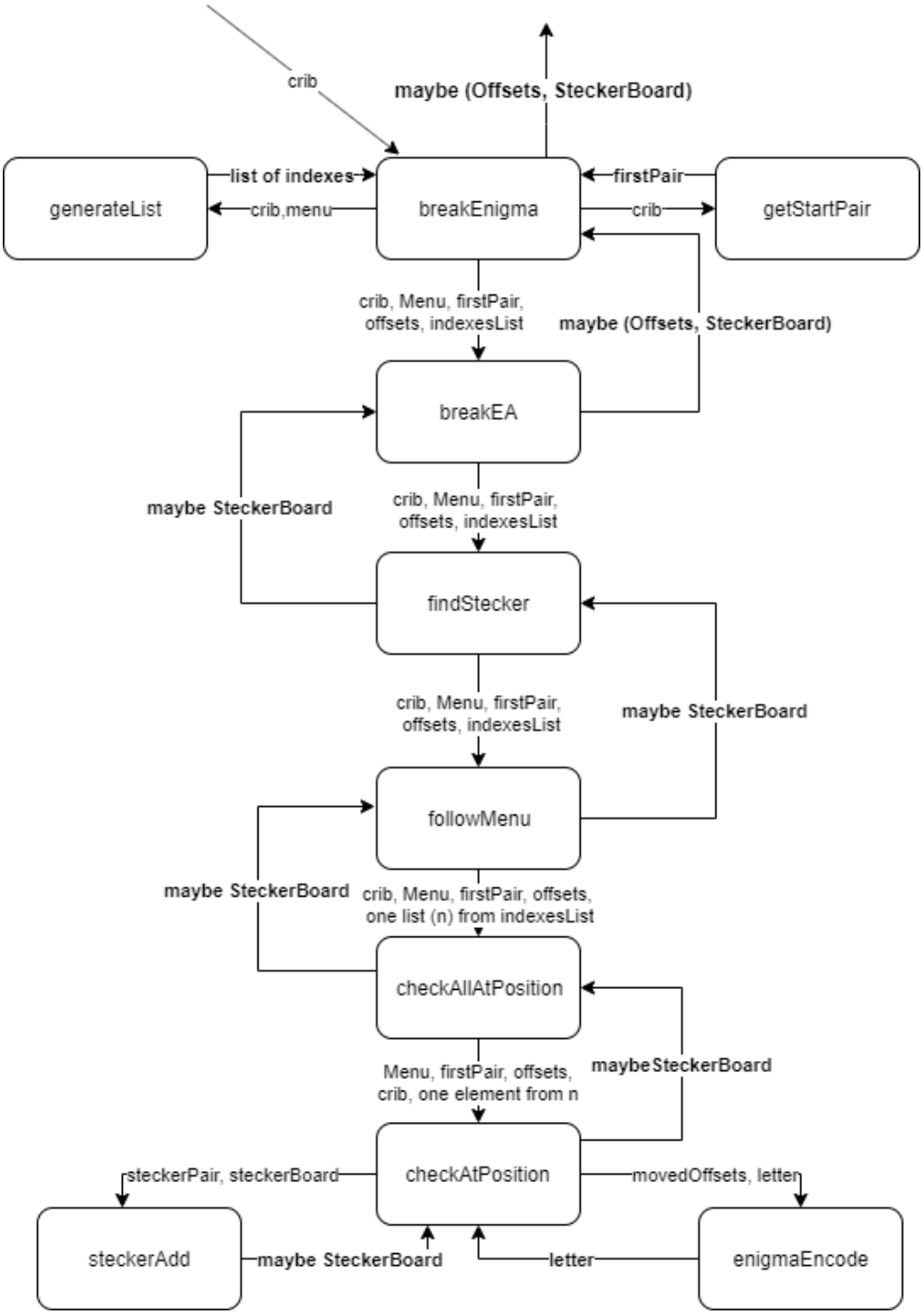
Instead of checking only 3 -> 6 -> 2 -> 7 -> 1, we would check 3 -> 5 -> 6 -> 2 -> 0 -> 4 -> 7 -> 1 ([[3,5],[6],[2,0,4],[7],[1]]]).

That will give us the results: Just ((0,6,4), [('R','C'), ('V','T'), ('U','N'), ('H','A'), ('B','M'), ('E','D')])

Which, used in enigmaEncodeMessage, give: "TTCMAANO".

```
Index:                  0 1 2 3 4 5 6 7
Ciphered:               T T C M A A N O
Result:                 T T C M A A N O
Correct matches:        + + + + + + + +
```

Initially, I implemented described algorithm by generating the list of indexes of letters at given index every time when calling followMenu. But after that I have realised that instead of doing that (it generates the same thing for each newPair in given offset - in the worst case it does that 406,250 (26*25*25*25) times!) I could just generate it once in enigmaEncode as the list of list and then pass is as an argument instead the longest menu. So my final implementation looked like that:



However, I quickly found out that this algorithm is extremely inefficient – to terminate with Nothing with this example:

p3="TURINGBOMBEHASKELLSIMULATIONSTOP"

x3="LKFMTWMTVKDEIVXHFHMNFDAZDRLMYQFRCKHHQSIMPIBZSXSCNMXVEKLXYRLEKZ"

It needed around 1.5h! Hence I've decided to completely change the approach.

I've realised that I don't need to check all those indexes every time I check every possible initial stecker pair and offsets. I can generate possible steckerboard in the same way as in initial Bombe and then I can validate it by checking for contraindications globally in the Crib.

My final implementation of improved algorithm works in very similar way to the initial Bombe. The only difference is in the findStecker function. Once the possible solution is generated, I check it by calling isUnique function. This function works like that:

1) It's looking for all instances of letters from generated stecker board in the plain text and it creates list with its indexes.
2) It calls followMenu with generated list as the menu to check for any contradictions.

I also wanted to check for any instances of those letters in cipher. However, it that case I couldn't simply repeat steps 1 and 2. Here's the reason why:

e.g. if we have steckerboard [('A','B'),('D','R')] and

p3="ADRK"

x3="LEWA"

We cannot call the followMenu in the same way I did in step 2, because we don't know if 'K' has a steckerpair or not. We don't have it in our steckerboard but it may mean that we haven't found it yet. However, there is very simple solution to this problem:

Once we've successfully done steps 1 and 2, we either found steckerPairs for 'L','E','W' or made sure that those letters don't have them. That's why we can

3) Generate similar list but with indexes of instances of letters from generated stecker board in the ciphered text

And then swap p3 with x3:

p3="LEWA"

x3="ADRK"

4) It calls followMenu with that list and new p3 and x3. We can do that because Enigma is inversible – i.e. if x is encoded to y, y will always be encoded to x and we already know all steckerpairs for p3.

That implementation works in almost the same time as the initial Bombe, but it produces much more verified solutions. However, I still didn't know 'how correct' my result was. That's why I added the last modification – my algorithm returns offsets, steckerboard and percent of certainty of correctness which is computed by dividing all checked elements in isCorrect by length of plain text.

To summarize I wanted to compare those 2 algorithms on a few examples:

p = "COMPUTERSCIENCECALIBRATIONSTRINGTESTINGONETWOTHREE"

x = "QWAVMZPNGFQVGWGYCKCXXHMEXTCGWPFOCWCSYXAEFUNXQFIZJW"

	Bombe	ExtendedBombe
Time	0.42 secs	0.43 secs
No of incorrect letters when encoded	5/50	0/50

p2="COMPUTERSCIENCESHEFFIELDUNIVERSITYSTOP"

x2="NAWLGAFTKDBKDLIKEOSZVAOKXKXFMQQUXBJDRCXRZFDKHLWCNBDJSMJBMJQCBBG"

	Bombe	ExtendedBombe
Time	0.02 secs	0.11 secs
No of incorrect letters when encoded	29/38	0/38

p3="COMPUTERSCIENCESHEFFIELDUNIVERSITYSTOP"

x3="FDANSQQXVLRHKURZDMWOLVQLZFXZJZXLVNZOJW"

Settings I've used: ((5,3,5),
[('P','M'),('I','N'),('K','F'),('L','O'),('S','D'),('U','A'),('B','V'),('H','R'),('E','C'),('G','T')]

	Bombe	ExtendedBombe
Time	0.02 secs	60 secs
No of incorrect letters when encoded	32/38	0/38
No of correct steckerpairs	0/10	10/10

As we can see, we are no longer forced to choose between correctness and efficiency. Even if we removed simplifications about offsets, the worst runtime would be around an hour. Also, ExtendedBombe may produce fully correct result even if the longestMenu is short (under condition that all letters from crib will be in the generated steckerboard). Finally, we can see how sure (in the worst case!) we can be about our solution.