Assessment 3

Question 2

Software Agents and Multi-Agent Systems

CS551K

Dr. Bruno Yun

Yassin Dinana

52094480

16th February 2021

one day extension

Yassin Dinana
52094480

Question 2:

A- In order to parse the given Aspartix files, a python solution is implemented that reads the file and converts it to the wanted representation. Two example files are given as a text file (.txt) and the other is with extension (.apx). Another file is included for testing where it includes 5 testing files with extension (.apx).

```
1   import re           #Regular Expression Module
2
3   examplesDirectory = "/Users/yassindinana/Desktop/University/Masters/Semester 2/Multi-Agents Systems/Assessments/Assessment 3/Question 2 " \
4                       "- Coding/ASMNTIII-Part2-files"
5   testsDirectory = "/Users/yassindinana/Desktop/University/Masters/Semester 2/Multi-Agents Systems/Assessments/Assessment 3/Question 2 - " \
6                    "Coding/ASMNTIII-Part2-files/tests/graphs"
7
8   nameExample_file = "/example.apx"
9   nameTest_file = "/test1.apx"
10  combiningDirectories = examplesDirectory + nameExample_file
11
12  def read_file(examplesDirectory):
13      dummy = open(examplesDirectory, "r")
14      Aspartix = dummy.read()
15      linesOfContent = Aspartix.split("\n")
16      extractedArguments = []
17      extractedAttacks = []
18      for line in linesOfContent:
19          if len(line) == 7:
20              extractedArguments.append(re.search(r"(?<=\().*?(?=\))", line).group(0))
21          elif len(line) > 7:
22              dummy = re.search(r"(?<=\().*?(?=\))", line).group(0)
23              dummy = dummy.split(",")
24              extractedAttacks.append(dummy)
25          else:
26                  dummy2=0
27
28
29      extractedArguments  = set(extractedArguments)
30      extractedAttacks    = [tuple(x) for x in extractedAttacks]
31      extractedAttacks    = set(extractedAttacks)
32      AF = (extractedArguments, extractedAttacks)
33      print(AF)
34      print(f"There are {len(extractedArguments)} arguments and {len(extractedAttacks)} attacks. ")
35      return AF
36
37
38  AF= read_file(combiningDirectories)
```

Figure 1: Python code for parsing the files and returning set of strings and pairs (code is used for parts A & B)

Looking at figure 1 above, the example files and the testing files are imported by adding their directories in the python program. The example files are then combined together as it can be seen in line 10.

Inside the function read_file, it can be seen that all the parsing steps can be found, the dummy function is used to convert categorical variables in the files. Two empty lists are then created which are given the names of extractedArguments and extractedAttacks, those lists are appended using the following for loop which reads the important information in the files, parses them, and appending the lists using the dummy function.

The needed arguments and attacks are then extracted from the files and the number of arguments and attacks is printed in the python console as an output.

The printing function uses the length of the strings and uses the read_file function and combined directories to print the output.

Yassin Dinana
52094480

B-

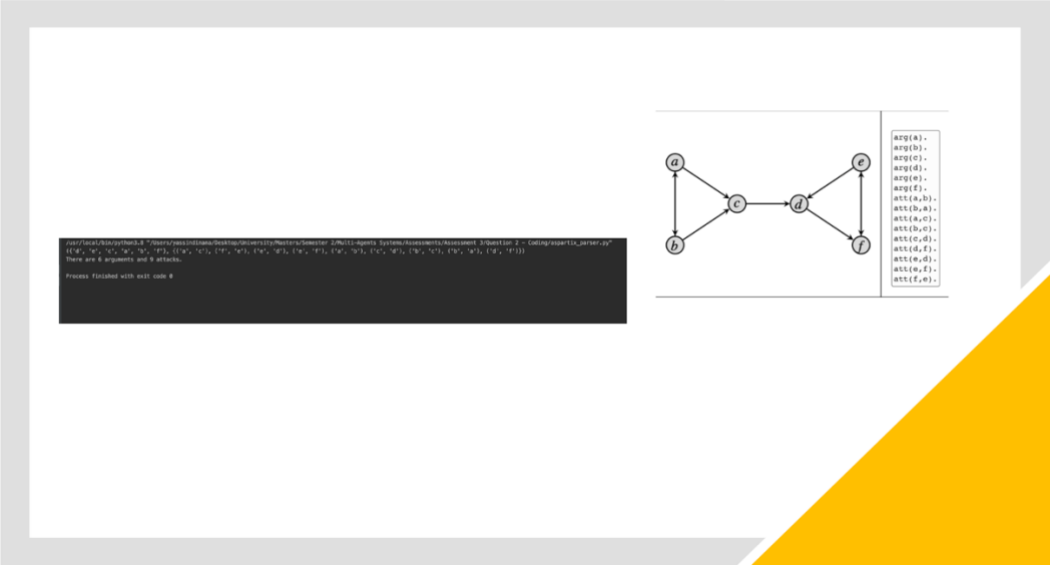

Figure 2: Output of the code in figure 1.

Looking at figure 2 below, the output of the code can be seen on the left side of the picture where it shows that there are 6 arguments and 9 attacks. The regular expression library is being used in the code in figure 1, where it helps match and find characters in a string. The python answer for this section is the same code in figure 1.

C-

```
def conflictFree(output):
    arguments = output[0]
    attacks = output[1]
    conflict_Free = []
    notConflict_Free = []
    for argumentX in arguments:
        for argumentY in arguments:
            temp = (argumentX, argumentY)
            temp1 = (argumentY, argumentX)
            if (temp in attacks) or (temp1 in attacks):
                notConflict_Free.append(set(temp))
            elif ((temp[0], temp[0]) in attacks) or ((temp[1], temp[1]) in attacks):
                dummy = 0
            else:
                conflict_Free.append(temp)

    conflict_Free = [list(element) for element in conflict_Free]
    conflictFreeFunc = []
    Remove = []

    for extension in conflict_Free:
        if extension[0] == extension[1]:
            conflictFreeFunc.append(extension[0])
        else:
            conflictFreeFunc.append(extension)
    for extension in conflictFreeFunc:
        for extensionR in conflictFreeFunc:
            if len(extension) > 1 and len(extensionR) > 1:
                if extension[0] == extensionR[1] and extension[1] == extensionR[0]:
                    if extension not in Remove:
                        Remove.append(extensionR)
```

Figure 3: Conflict-Free Algorithm

In order to compute the maximal using the conflict-free method, by looking at the figure 2, it can be seen that not all nodes attack or defend each other against other nodes. The conflict-free algorithm will show which nodes are conflict free and do no interact with each other. The algorithm is implemented in python as a function.

When looking at the conflict-free function, the algorithm searches in the files for all the nodes that do not interact with each other, in order to do that, two different lists are created were one list named “conflict_free” will include all the conflict free nodes and all other nodes who interact with other nodes by attacking or defending will be stored in another list called “notConflict_Free”.

A For-loop with an if statement is taking place in order to append the lists created when there is / not conflict_free nodes. The extension is available in the code which allows the string to be updated regularly by adding or removing new nodes from the list.