6COSC020W.1 Applied AI

Tutorial Week 9

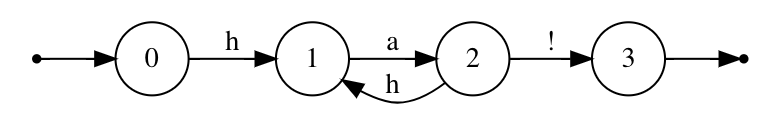
Dr Maria Chondrogianni

**Natural Language Processing**

**Sample answers**

1. Write a Finite State Automaton able to laugh, i.e. to write on your computer screen ha!, haha! Hahaha!

Sample answer (this uses points to make start and terminal states – this and the next solution created using Graphviz):

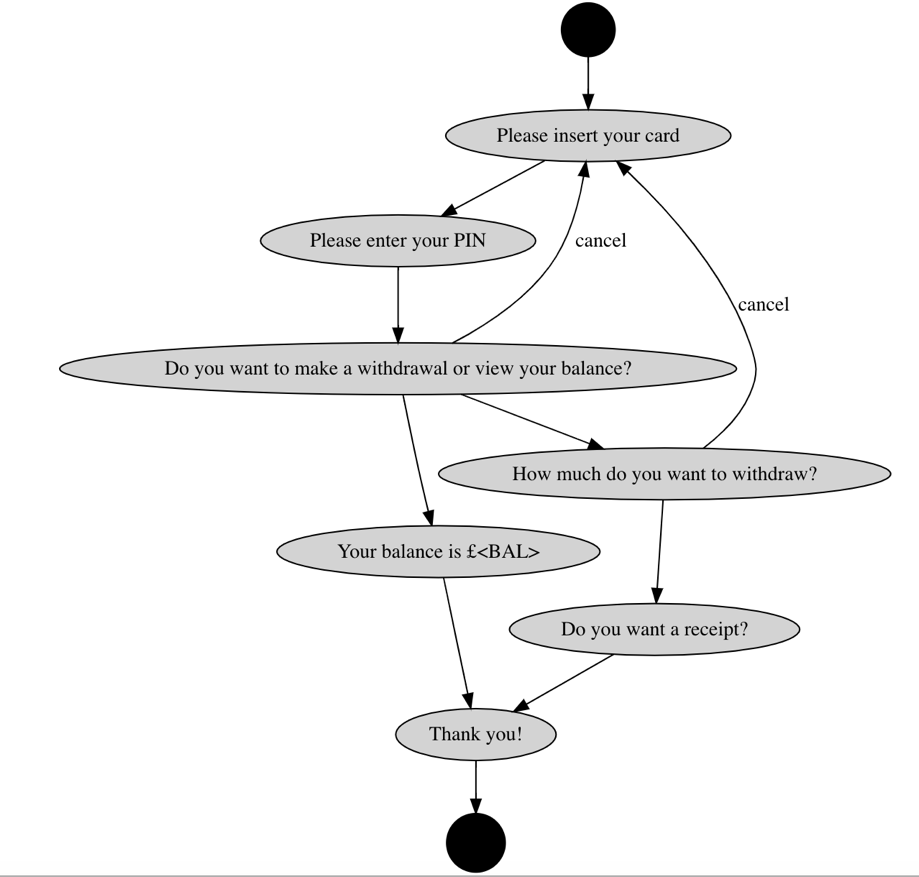


This is one possibility. Students can start from initial state 0 or 1 and finish at state 3 or at state 4. The FSA can go back at state 2 (or 3) by writing h or it can go back to initial state by writing nothing.

1. A bank wants to develop an automated customer assistant, available via kiosks in its branches.  The automated assistant will allow customers to make requests of the assistant, using a simple question-answer dialog.  The assistant can display the balance of a customer account, make a withdrawal from the account, print a statement etc.

Using a finite state automaton, design a simple dialogue manager for the automated bank assistant.  The dialog manager should show the different stages in the dialog from the assistant’s perspective.  Hint: the assistant should first ask the customer to verify themselves using the same kind of procedure you would use at a normal ATM.  The assistant should also allow the customer to cancel and return to the beginning of the dialog.

Sample solution – sample code follows on the following page. Note that you will need Python 3.



|  |
| --- |
| import re |
|  |  |
|  | class Node : |
|  |  |
|  | def \_\_init\_\_(self,m): |
|  | self.message = m; |
|  | self.edges = [] |
|  |  |
|  | def next(self) : |
|  | while (True) : |
|  | response = input(self.message + "\n") |
|  | if len(self.edges) == 0 : # end of dialog |
|  | return None |
|  | # find which action to execute |
|  | for e in self.edges : |
|  | if e.match(response) : |
|  | return e.getNextNode() |
|  | print("Sorry, I don't understand. Please try again.\n"); |
|  |  |
|  | def addEdge(self,edge) : |
|  | self.edges.append(edge) |
|  |  |
|  | def getMessage(self) : |
|  | return self.message |
|  |  |
|  | class Edge : |
|  |  |
|  | def \_\_init\_\_(self,pattern,n) : |
|  | # pattern is a regular expression representing the |
|  | # pattern of expected user input |
|  | self.regexp = re.compile(pattern) |
|  | self.nextNode = n |
|  |  |
|  | def getNextNode(self) : |
|  | return self.nextNode |
|  |  |
|  | def match(self,r) : |
|  | # Match a given string against the expected response |
|  | # using regular expressions |
|  | if self.regexp.match(r) : |
|  | return True |
|  | else : |
|  | return False |
|  |  |
|  |  |
|  | def run() : |
|  | # Create nodes for the different stages of the dialog |
|  | start = Node("Please enter your card") |
|  | pin = Node("Please enter your pin") |
|  | acttype = Node("Do you want to make a credit, a withdrawal or view your balance?") |
|  | credit = Node("How much do you want to credit?") |
|  | withdraw = Node("How much do you want to withdraw?") |
|  | bal = Node("Here is your balance?") |
|  | thankyou = Node("Thank you") |
|  |  |
|  | # Now connect the nodes using edges which state what user input |
|  | # is needed to reach the next node. We use regular expressions |
|  | # to represent user input |
|  | start.addEdge(Edge("[0-9]{16}",pin)) # 16-digit card number |
|  | pin.addEdge(Edge("[0-9]{4}",acttype)) # 4-digit pin number |
|  | # here we could go in two directions, depending on what the user says |
|  | acttype.addEdge(Edge("withdraw",withdraw)) # note this is a regexp constant |
|  | acttype.addEdge(Edge("credit",credit)) |
|  | acttype.addEdge(Edge("view balance",bal)) |
|  | credit.addEdge(Edge("[0-9]+\.[0-9]\*",thankyou)) |
|  | withdraw.addEdge(Edge("0-9]+\.[0-9]\*",thankyou)) |
|  | bal.addEdge(Edge("",thankyou)) # we assume any user input will take us to the last step |
|  |  |
|  | node = start.next() |
|  | while (node != None) : |
|  | node = node.next() |
|  | print("Goodbye") |
|  |  |
|  | run() |

1. If you have a smart phone with you, do try to test your voice assistant’s abilities by answering the following questions (you can work individually or in small groups):
2. How many ways can you ask your voice assistant to send a message to one of your contacts (that it understands)?

Sample answers

--------------

Send X a whatsapp message to say hello

Say hello to X with a whatsapp message

Say hello using whatsapp to X

1. Try asking your voice assistant "How long would it take me to walk to the nearest McDonalds?" Does your voice assistant answer this question correctly - e.g., with a time?  Why do you think it answers the way it does?

(Siri finds list of nearby McDonalds and asks which one you want. Location only pattern matching applies.)

1. Ask your voice assistant what the weather will be on Saturday.

Now ask if you will need an umbrella on Saturday. Does the voice assistant answer both questions correctly? (Siri does)

1. Ask your voice assistant the following "I really want to read a Harry Potter book.  Where can I buy one?"

What happens when you ask this question?  Why do you think the voice assistant answers the way it does? (Siri finds entries on the web about reading Harry Potter books. The two utterances cannot get combined)

1. Ask your voice assistant the following:

 1. "What's the best route, if I'm driving, to get to Liverpool\*?"

2. "What's the best way, if I'm going by car, to get to Liverpool\*?"

   Do you get the same results in exactly the same way?  (Siri opens different apps, as it cannot cope with embedded sentences). Can you find a way to ask this question that confuses your voice assistant?

1. Using your voice assistant try to find the cheapest way to get to Oxford\* using public transport.  How easy is it? (‘Public transport’ does not always get matched to bus, train etc. This is a semantics problem.)

\*You can adapt locations if you prefer.

-

1. Find further information on GPT-3 and write a short paragraph of approximately 100 words with the outcomes of your research. Do not forget to include the sources (references) that you used for your paragraph.