10.009 The Digital World

Term 3. 2020

Problem Set 9 - Part I (for Week 9)

Last update: January 9, 2020

Due dates:

- **Problems: Cohort sessions**: Following week: Tuesday 11:59pm.
- **Problems: Homework**: Same as for the cohort session problems.
- **Problems: Exercises**: These are practice problems and will not be graded. You are encouraged to solve these to enhance your programming skills. Being able to solve these problems can help you prepare for the final examination.

Objectives:

1. Create state machines from given state diagrams of time step tables.
2. Use SM class to run state machine.
3. Draw state transition diagrams.
4. Write time-step tables.

Note: Solve the programming problems listed below using any of your preferred editor. Make sure you save your programs in files with suitably chosen names and in an newly created directory. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Note: This handout only contains the questions for Problem Set 9 - Part I. Questions for Problem Set 9 - Part II can be found in eDimension.

Problems: Cohort sessions

1. *State Machine: Coke:* In this problem, you will implement in Python the behavior of a simplified coke-dispensing machine. The behavior of such a machine is captured in the state diagram shown in Figure 1. The machine consists of two states labelled 0 and 1. The state diagram does not show what the machine would do if an unexpected coin is inserted. Therefore, assume that any unexpected coin is returned to the user without a change in the machine's state. Thus, on your own, you may want to complete Figure 1 to add in the missing transitions.

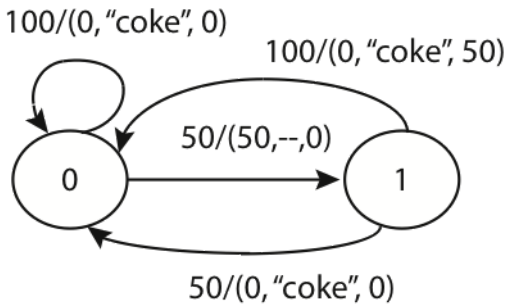

Figure 1: State diagram for a simple coke dispenser machine.

Each directed arc in the state diagram is labelled as `x/y` where x denotes the input received and y, the output generated. For example, the arc that connects state 0 to state 1 that's labelled `50/(50, '--',0)` means that when the dispenser receives 50¢ (50 before the /) in state 0 it moves to state 1 and generates an output of `(50, '--',0)`. This tuple of values in the output indicates that the dispenser display shows 50 which is the amount entered by the user, no coke has been dispensed yet as indicated by `'--'`, and no change has been returned to the user as indicated by the last entry which is a 0.

The machine accepts only 50¢ and one dollar (100¢) coins. It has a display that shows how many cents have been deposited.

- State 0: When a 50¢ coin is deposited the dispenser moves to state 1. At this moment in time, the display shows 50 but nothing is dispensed and no change is returned. If a dollar coin is deposited, the machine continues to display 0, dispenses coke, and does not return any money (well, why should it!).
- State 1: When a 50¢ coin is deposited the dispenser moves to state 0. At this moment in time, the display shows 0, coke is dispensed and no change is returned. If a dollar coin is deposited the machine continues to display 0, dispenses coke, and returns 50¢.

Python program: We wish to write a Python program that simulates the behavior of

the coke dispenser as described above. We will write a class named CM that contains attributes and methods as described below:

- CM class is a subclass of sm.SM class, which is part of the libdw module.
- CM class has a class attribute called `start_state` which is the starting state of the machine. This attribute should be initialized to 0, which represents state 0 in the diagram above.
- CM class has a method named `get_next_values(self, state, inp)` that takes in the current state and the input, and returns the next state and output as a tuple. Think about the following: which state represents the following scenarios?
  (a) the coke machine is waiting for a valid coin to be deposited
  (b) the coke machine has a 50-cent coin in it

Sample interaction:

```
>>>
>>> c=CM()
>>> c.start()
>>> c.step(50)
(50, '--', 0)
>>> c.step(50)
(0, 'coke', 0)
>>> c.step(100)
(0, 'coke', 0)
>>> c.step(10)
(0, '--', 10)
>>> c.step(50)
(50, '--', 0)
>>> c.step(100)
(0, 'coke', 50)
>>> c.step(10)
(0, '--', 10)
```

Submission to Vocareum: Please submit your entire class with the `start_state` and the `get_next_values` defined.

2. *State Machine: SimpleAccount:* In this problem, you will need to create a state machine that simulates a simple bank account. This is similar to the Accumulator state machine

in the text book. The only difference is that any withdrawal when the balance is less than $100 incurs a $5 charge. The state machine should fulfill the following:

- The starting balance is specified when instantiating the object.
- The output of the state machine is the current balance after the transaction.

Sample interaction:

```
>>> acct=SimpleAccount(110)
>>> acct.start()
>>> acct.step(10)
120
>>> acct.step(-25)
95
>>> acct.step(-10)
80
>>> acct.step (-5)
70
>>> acct.step(20)
90
>>> acct.step(20)
110
```

Think about the following: How is this state machine different from the one in Question 1?

Problems: Homeworks

1. *State Machine: Comments:* Write a state machine whose inputs are the characters of a string. The string contains the code for a Python program. The output of the state machine are either (a) the input character if it is part of a comment or (b) `None`. As you know, comments start with a `'#'` character and continue to the end of the current line. In Python, if you want to create a string that contains new line character, you can use `'\n'`.

   For example:

```
>>> str = 'def f(x): # comment\n   return 1'
>>> m = CommentsSM()
>>> m.transduce(str)
[None, None, None, None, None, None, None, None, None, None,
'#', ' ', 'c', 'o', 'm', 'm', 'e', 'n', 't',
None, None, None, None, None, None, None, None, None, None, None,
    None]
```

   Note: The newline character is `'\n'`, so you can test for the end of the line with:

```
if inp == '\n':
```

   You should start by drawing a state transition diagram indicating the states and what inputs cause transition to which other states. Use the test case above to determine if your state transition diagram is correct. You should begin writing your program only when you are confident that your diagram is correct.

   We encourage you to debug this on your machine in an IDE. Open a file in an IDE, enter your class definition and testing examples in that file, and do Run Module to execute.

2. *State Machine: First Word:* Write a state machine whose inputs are the characters of a string and which outputs either (a) the input character if it is part of the first word on a line or (b) None. For the purposes here, a word is any sequence of consecutive characters that does not contain spaces or end-of-line characters. In this problem, comments have no special status, if the line begins with `'# '`, then the first word is `'#'`.

   For example:

```
>>> str = 'def f(x): # comment\n   return 1'
>>> m = FirstWordSM()
>>> m.transduce(str)
['d', 'e', 'f',
None, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, None, None,
'r', 'e', 't', 'u', 'r', 'n',
None, None]
>>>
```

   Note: The newline character is `'\n'`, so you can test for the end of the line with:

```
if inp == '\n':
```

Use the test case above to determine if your state transition diagram is correct. You should begin writing your program only when you are confident that your diagram is correct. We encourage you to debug this on your machine in an IDE.

Access eDimension for the rest of Problem Set 9.