CSE 318/409 Programming Project

Due: Wednesday May 9 at 2PM

Submitted through course site.

This is an **individual** project.

This document was last updated: 12:31PM; 4/18/2018. Change log in last slide

WARNING

- **Under no circumstances copy code from classmates or elsewhere in full or in part**
- The instructor has already set a program to identify similar code
- If the program flags submission(s), regardless if it is the complete or partial copy:
  - The instructor will examine the submission(s)
  - If the instructor deems it a copy, the student(s) will receive an NF in the course and the case will be brought before the Disciplinary Committee
  - The student will not hear from the instructor; instead the student will receive a summon to appear before the Disciplinary Committee for a hearing
- **Every case brought by the instructor to the Disciplinary Committee has resulted in an F as grade for the course assigned to the student**
- **Most cases brought by the instructor has resulted in immediate suspension of the student for a whole semester**
  - That is, the student would immediately be suspended and only be allowed to return until the Spring 2019

Eligibility

You should have received an email indicating if you are exempt from the final exam or not:

- Let Score = (Test # 1 + Test # 2) / 2
- If Score < 70% then student must do the final exam and programing project gives extra credit
    - Consider doing a subset of the programming tasks as opposed to the whole project. But please keep all headings from functions you do not work on.
- If Score ≥ 70% them student may do the programming project as a substitute for the final exam. In this case the final exam will give the student extra credit

Restrictions

- Your program must run on **Python 3.6.4**
- **No software libraries/packages (e.g., using #include)** may be used unless explicitly authorized by the instructor via email
  - You must give a good reason.
  - Nevertheless expect the instructor to say “no”

Inputs and submission

- InputFile318Skeleton.py: a skeleton file with the classes and functions. You must complete the functions in that same file. **Don’t change/delete the function headings/parameters**.
- TestFile318.py: a sample test file; exact same format as the one I will use when testing your program.

- **Submission**: a single file, InputFile318YourName.py. For example, InputFile318YMunoz.py. Submitted through course site.

The input file

- Don’t edit/delete any of the class definitions and headings
- Add your name and list of functions you are submitting as a comment (see Line 4 of the inputFile318Skeleton.py):

```
### NAME: put your name here
### LIST: please list all functions you are submitting for grading here:
### ...list functions here
### Any function you are not submitting for grading please leave
### it unmodified, as it was originally in this file. DO NOT DELETE IT
```

Four Parts

- DFAs
- PDAs
- TMs
- NTMs

DFAs: Input Format

```
dfa1 = DFA(
    {'q0','q1'}, # Q
    {'0','1'}, # Sigma
    {'q0':{'0':'q0', '1': 'q1'},
     'q1':{'0':'q1', '1': 'q0'}}, # Delta
    'q0', # q0
    {'q1'} # F
    )
```

Don’t change format!

DFAs: Class

```
class DFA(object):
    def __init__(self, Q=None, Sigma=None, Delta=None, q0=None, F=None):
        self.Q = Q
        self.Sigma = Sigma
        self.Delta = Delta
        self.q0 = q0
        self.F = F
```

Don’t change format!

DFAs: Verify$_{\text{DFA}}$

```
def verifyDFA(self):
        # Decides if self is a correct DFA
        # Verification needs to be extended.
        # DO NOT change the input representation of DFAs
```

Don’t change/delete function heading!

DFAs: Accept$_{DFA}$

```
def acceptDFA(self,s):
    # Decides if self  accepts s
```

Don’t change/delete function heading!

DFAs: $\text{Empty}_{\text{DFA}}$

```
def emptyDFA(self):
        # Decides if self  accepts no  strings
```

Don’t change/delete function heading!

DFAs: EQ_{DFA}

```
def EQDFA(self,D):
    # Decides if L(self) = L(D), where D is a DFA
```

Don’t change/delete function heading!

PDAs: Input Format

```
pda1 = PDA(
    {'q1','q2','q3','q4'}, # Q
    {'0','1'}, # Sigma; may not contain 'e', which we use to denote the empty string
    {'0','$'}, # Gamma; may not contain 'e', which we use to denote the empty string
    {'q1':{'e': [['q2','e','$']]},
     'q2':{'0': [['q2','e','0']], '1': [['q3','0','e']]},
     'q3':{'1': [['q3','0','e']], 'e': [['q4','$','e']]},
     'q4':{}
    }, # Delta
    'q1', # q0
    {'q1','q4'} # F
    )
```

Don’t change format!

PDAs: Class

```
class PDA(object):
    def __init__(self, Q=None, Sigma=None, Gamma=None, Delta=None, q0=None,
F=None):
        self.Q = Q
        self.Sigma = Sigma # may not contain 'e', which we use to denote the empty string
        self.Gamma = Gamma # may not contain 'e', which we use to denote the empty string
        self.Delta = Delta # may use 'e', which we use to denote the empty string
        self.q0 = q0
        self.F = F
```

Don’t change/delete class definition!

PDAs: Verify$_{\text{PDA}}$

```
def verifyPDA(self):
        # Decides if self is a correct PDA
        # 'e' denotes the empty string
        # DO NOT change the input representation of PDAs
```

Don’t change/delete function heading!

PDAs: Accept$_{\text{PDA}}$

```
 def acceptPDA(self,s):
        # Decides if self accepts s with at most 2|s| transitions from the
start state.
        # Must try all possible transitions
```

Don’t change/delete function heading!

PDAs: NEQ$_{\text{PDA}}$

```
 def notEQPDA(self,P,k):
        # Quasi-recognizes if L(self) != L(P), where P is a  PDA
        # try all strings of length 0, 1, 2, .., k.
        # When it reaches strings of length k+1, it returns false.
        # A true recognizer would not stop at k and simply continue
running.
        # We add k so it always terminate
```

Don’t change/delete function heading!

TMs: Input Format

```
tm1 = TM(
    {'q0','q1','accept','reject'}, # Q
    {'0','1'}, # Sigma
    {'0','1','_'}, # Gamma; '_" is the symbol that must be used for blank
    {'q0':{'0': ['q0','0','R'], '1': ['q1','1','R'], '_': ['reject','_','L']},
     'q1':{'0': ['q1','0','R'], '1': ['q0','1','R'], '_': ['accept','_','L']},
    }, # Delta
    'q0', # q0
    'accept', #qAccept
    'reject' #qReject
    )
```

Don’t change format!

TMs: Class

```
class TM(object):
    def __init__(self, Q=None, Sigma=None, Gamma=None, Delta=None, q0=None, qAccept=None, qReject=None):
        self.Q = Q
        self.Sigma = Sigma
        self.Gamma = Gamma # '_' is the blank symbol
        self.Delta = Delta # Move to left: 'L'; move to right: 'R'
        self.q0 = q0
        self.qAccept = qAccept
        self.qReject = qReject
```

Don’t change format!

TMs: Verify$_{TM}$

```
def verifyTM(self):
        # Decides if self is a correct TM
        # Verification needs to be extended.
        # '_' is the blank symbol
        # DO NOT change the representation of TM
```

Don’t change/delete function heading!

TMs: Accept$_{\text{TM}}$

```
def acceptTM(self,s,k):
    # Quasi-recognizes if TM self accepts s
    # if TM reaches an accepting state, it should accept
    # if TM reaches a rejecting states it should reject.
    # '_' is the blank symbol
    # s is a list; it is used as the initial tape;
    # assumes head pointing to first symbol in s
    # returns false if the number of transitions exceeds k.
    # A true recognizer would not stop at k and simply continue running.
    # We add k so it always terminate
```

Don’t change/delete function heading!

NTMs: Input Format

```
ntm1 = NTM(
    {'q0','q1','accept'}, # Q
    {'0','1'}, # Sigma; may not contain 'e', which we use to denote the empty string.  May not contain '_', which we use for the blank symbol
    {'0','1','_'}, # Gamma; '_" is the symbol that must be used for blank; may not contain 'e', which we use to denote the empty string
    {'q0':{'0': [['q0','0','R']], '1': [['q1','1','R']]},
     'q1':{'0': [['q1','0','R']], '1': [['q0','1','R']], '_': [['accept','_','L']]},
    }, # Delta
    'q0', # q0
    'accept', #qAccept
    )
```

Don’t change format!

NTMs: Class

```
class NTM(object):
    def __init__(self, Q=None, Sigma=None, Gamma=None, Delta=None, q0=None, qAccept=None):
        self.Q = Q
        self.Sigma = Sigma
        self.Gamma = Gamma # '_' is the blank symbol
        self.Delta = Delta # Move to left: 'L'; move to right: 'R'
        self.q0 = q0
        self.qAccept = qAccept
```

Don’t change format!

NTMs: Verify$_{\text{NTM}}$

```
def verifyNTM(self):
        # Decides if self is a correct NTM
        # 'e' denotes the empty string; '_' is the blank symbol
        # DO NOT change the representation of NTM
```

Don’t change/delete function heading!

NTMs: Accept$_{\text{NTM}}$

```
def acceptNTM(self,s,k):
        # Quasi-recognizes if NTM self accepts s; NTM has no reject state
        # s is a list; it is used as the initial tape;
        # assumes head pointing to first symbol in s
        # If it doesn't reach the accepting state with all transitions of length k, return false.
        # A true recognizer would not stop at k and simply continue running.
        # We add k so it always terminate
```

Don’t change/delete function heading!

Questions

- You can use Piazza for questions about the project
- I will answer questions regarding the format of the inputs
- I will answer clarification questions about what the various functions are suppose to do
- Likely I will not be able to answer questions regarding Python
  - But do post them as classmates may be willing to help

Change Log

- 12:31PM; 4/18/2018: remove a comment in Slide 25