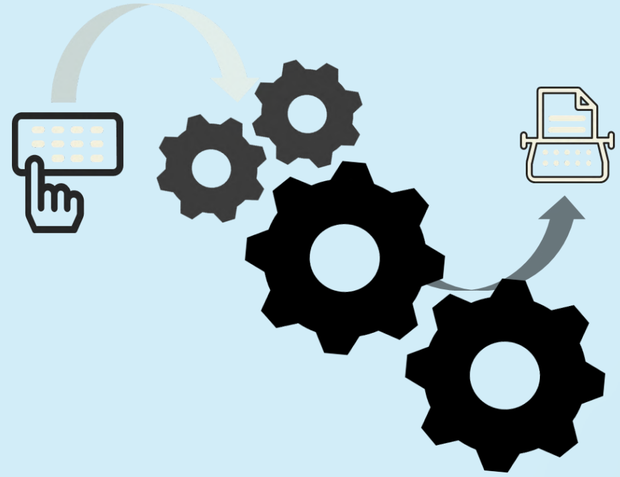


21AIE301 FORMAL LANGUAGES AND AUTOMATA

Morphological Generator

TEAM -10

TEAM MEMBERS

M.BHOOMIKA
20008

K.M.KRISHNAN
20031

M.SPANDANA
20038

R.THUSHIT
20072

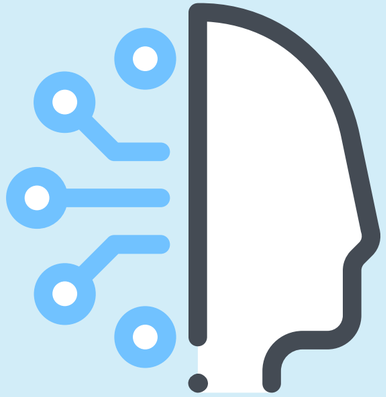
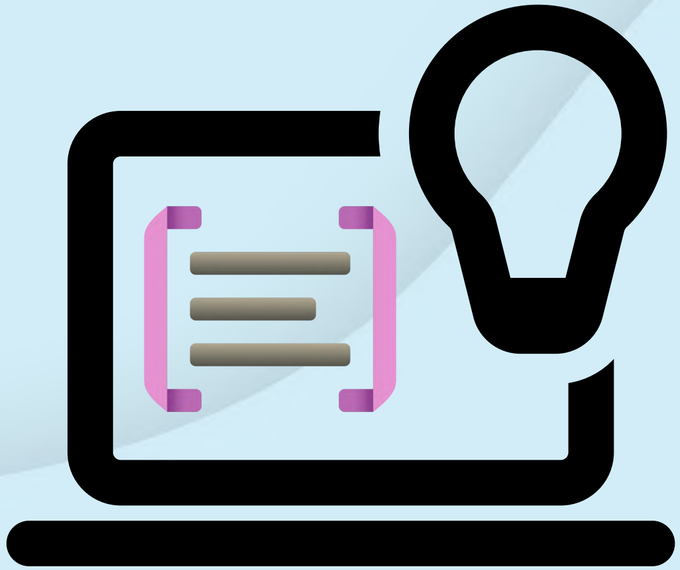


Table of contents

MORPHOLOGY GENERATION



Introduction

Morphology → The area of computational linguistics that deals with the structure of words.
→ Uses Finite State Technology

Finite State Automata :

Captures the rules of many languages ; used for word generation using morphemes.

Finite State Transducers :

maps the surface form of a word to a description of the morphemes that constitute that word or vice versa.

Morphological Operation

INFLECTION

Creates different forms of the same word

Example : cat → cats , play → playing

DERIVATION

Creates different words from the same lemma

Example : Success → successful → unsuccessful → unsuccessfully

COMPOUNDING

Combines two words into a new word

Example: Home + work → Homework

CLITICIZATION

A morpheme that acts like a word but is reduced

Example : Did not → didn’t

Morphological Operation

INFLECTION

Creates different forms of the same word

Example : cat → cats , play → playing

DERIVATION

Creates different words from the same lemma

Example : Success → successful → unsuccessful → unsuccessfully

COMPOUNDING

Combines two words into a new word

Example: Home + work → Homework

CLITICIZATION

A morpheme that acts like a word but is reduced

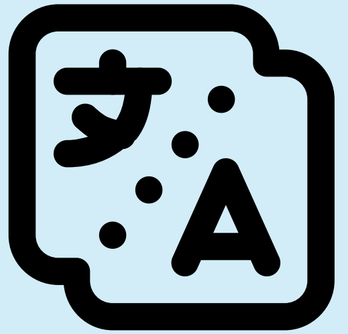
Example : Did not → didn’t

Inflection - Verbs

WORD FORMS	EXAMPLE
Infinitive / present tense	walk, go
Singular present tense	walks, goes
Simple past	walked, went
Past Participle (ed-form)	walked, gone
Present participle (ing-form)	walking, going

Inflection - Nouns

WORD FORMS	EXAMPLE
Number	singular (book) vs plural (books)
Possessive	book’s, books
Personal pronouns inflect for person, number, gender, case:	I saw him; he saw me; you saw her; we saw them; they saw us



Finite State Automata and Finite State Transducer

Finite State Automata

Simplest machine to recognize patterns

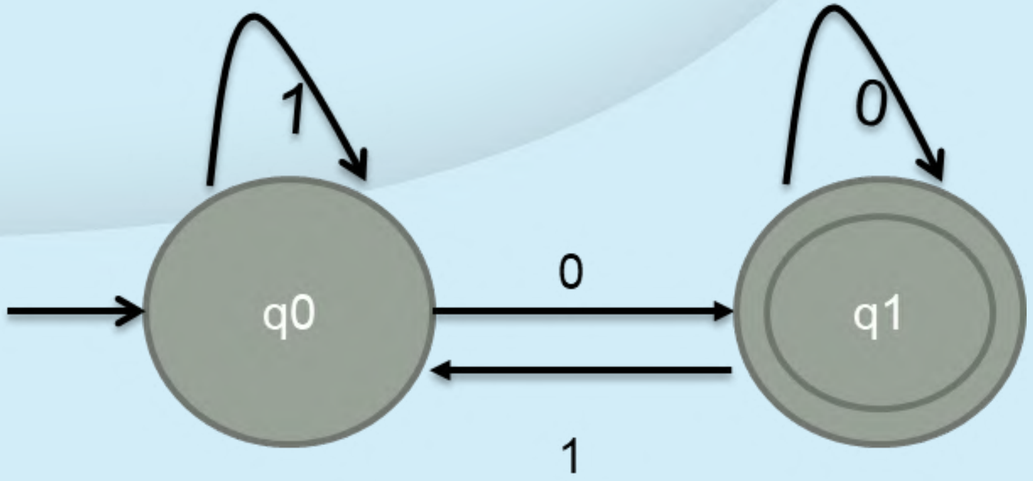
Abstract machine that has **5 tuples**

Possesses **a set of states & rules to move from one state to another** (depending on input symbol)

Two types: DFA & NFA

Q : Finite set of states
Σ : set of Input Symbols
q : Initial state
F : set of Final States
δ : Transition Function.

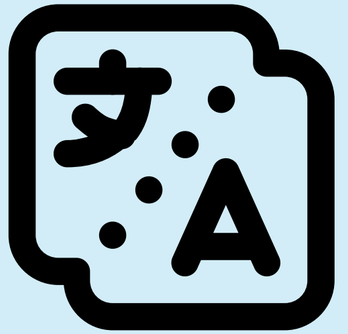
EXAMPLE



Below DFA with Σ = {0, 1} accepts all strings ending with 0

Initial state :q0
Final state:q1

Finite State Transducer



FSA that **produces Output as well as reads input.**

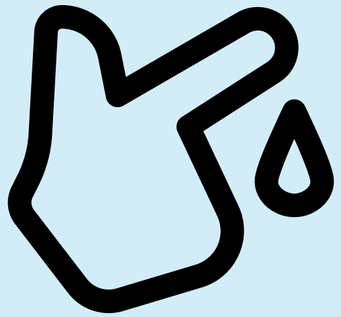
→ FST : Parsing || FSA: Recognizing

Consists of **finite number of states** linked by transition labelled with **input output** pair.

Application: NLP and Speech recognition

Abstract machine that has **6 tuples**

"Q : Finite set of states "
"Σ : set of Input Symbols"
"Γ":set of Output Symbols
"q : Initial state"
"F : set of Final States"
"δ : Transition Function."



EXAMPLE

$$T = L\_in \times L\_out$$

defines a relation between two regular languages L_in and L_out

L_in={cat, cats , fish , fishes ,....}
l_out={cat+N+sg, cat+N+pl, fish+N+sg,fish+N+pl}

T= { <cat,cat+N+sg>,
<cats,cat+N+pl>,
<fish,fish+N+sg>,
<fishes,fish+N+pl>
}

Aim

1) DESIGNING A FSM THAT ACCEPTS ALL FORMS THE WORDS IN A GIVEN CORPUS
2) DESIGNING A CODING AND CODING A NON-DETERMINISTIC FST TO CONVERT INFINITE WORD TO ITS CONTINOUS FORM

CORPUS

The cheerful sun shone brightly in the sky, casting its warm rays on the Earth below. The birds sang joyfully as they flew through the air, their melodies filling the air with happiness. The flowers bloomed, their petals opening up to soak in the sun's warm embrace. The bees buzzed from flower to flower, collecting nectar and pollen to bring back to their hive. The people outside basked in the sun's warmth, enjoying the beautiful day.

Verbs: shone, cast, sang, flew, bloomed, opened, soaked, buzzed, collecting, basked, enjoying

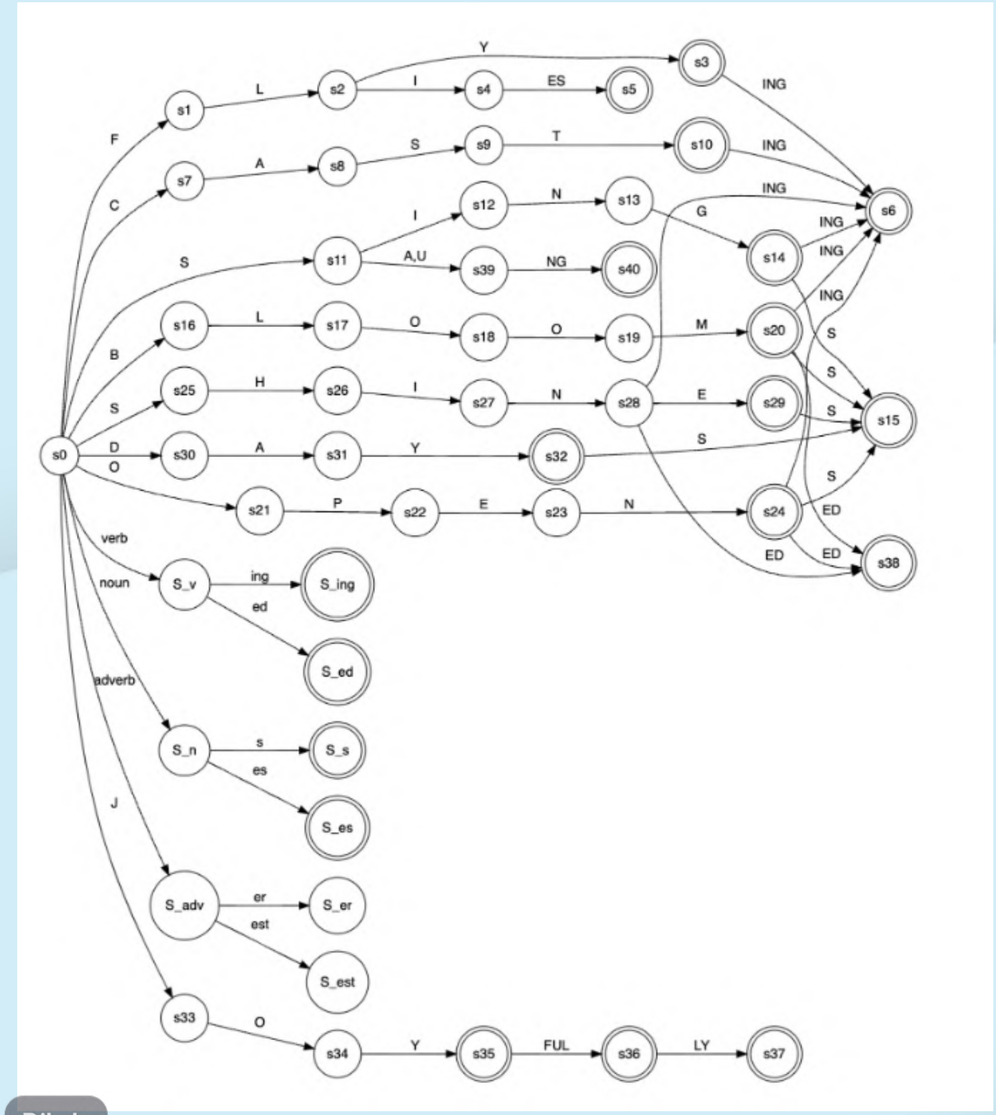
Nouns: sun, sky, Earth, rays, birds, air, melodies, flowers, petals, sun's warmth, bees, nectar, pollen, hive, people, day

Adverbs: cheerfully, brightly, joyfully, warmly

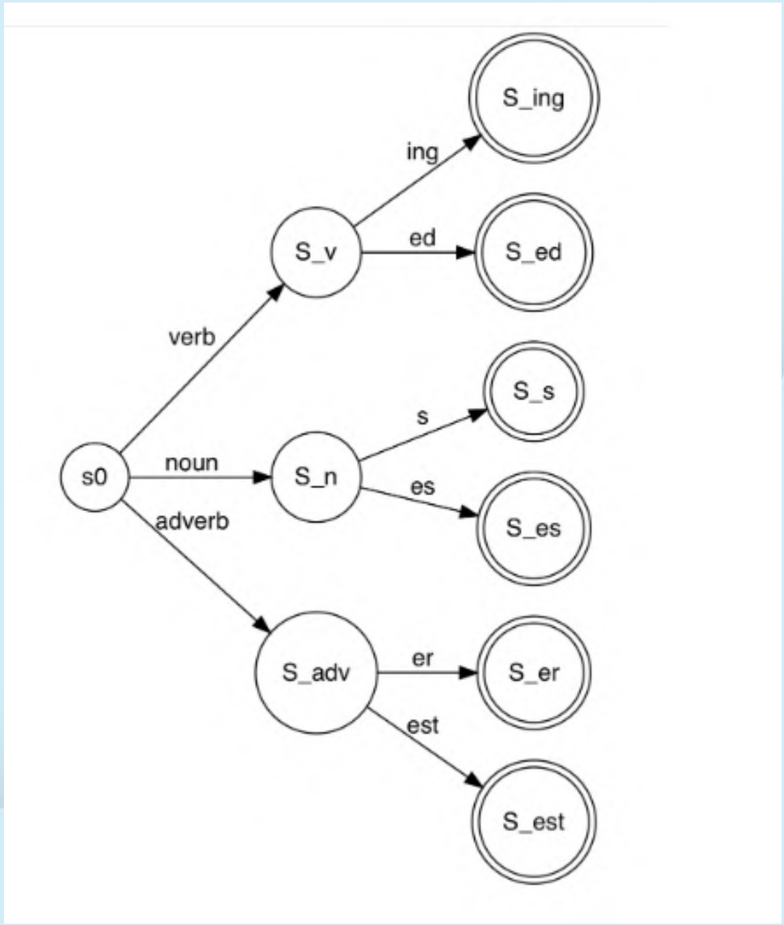
{Fly , Cast , Sing , bloom , open ,collect , shine , joy , day}

FLY	FLIES , FLYING , FLEW
CAST	CASTING
SING	SINGS , SINGING, SANG, SUNG
BLOOM	BLOOMS, BLOOMING , BLOMMED
OPEN	OPENS, OPENING, OPENED
COLLECT	COLLECTS, COLLECTING , COLLECTED
SHINE	SHINES, SHINING , SHONE
JOY	JOYFUL , JOFULLY
DAY	DAYS

Non-Deterministic finite automata



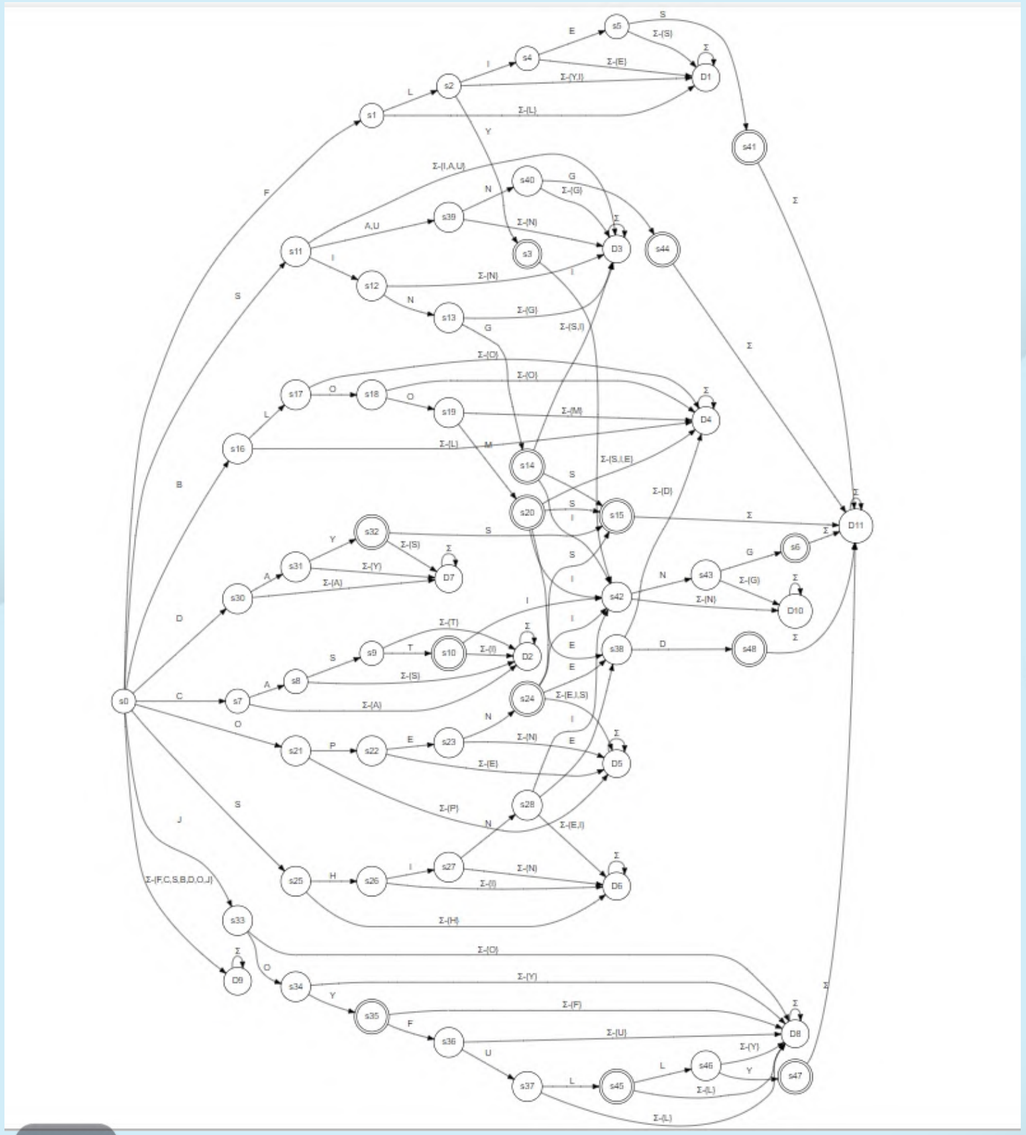
NFA Generalised Diagram



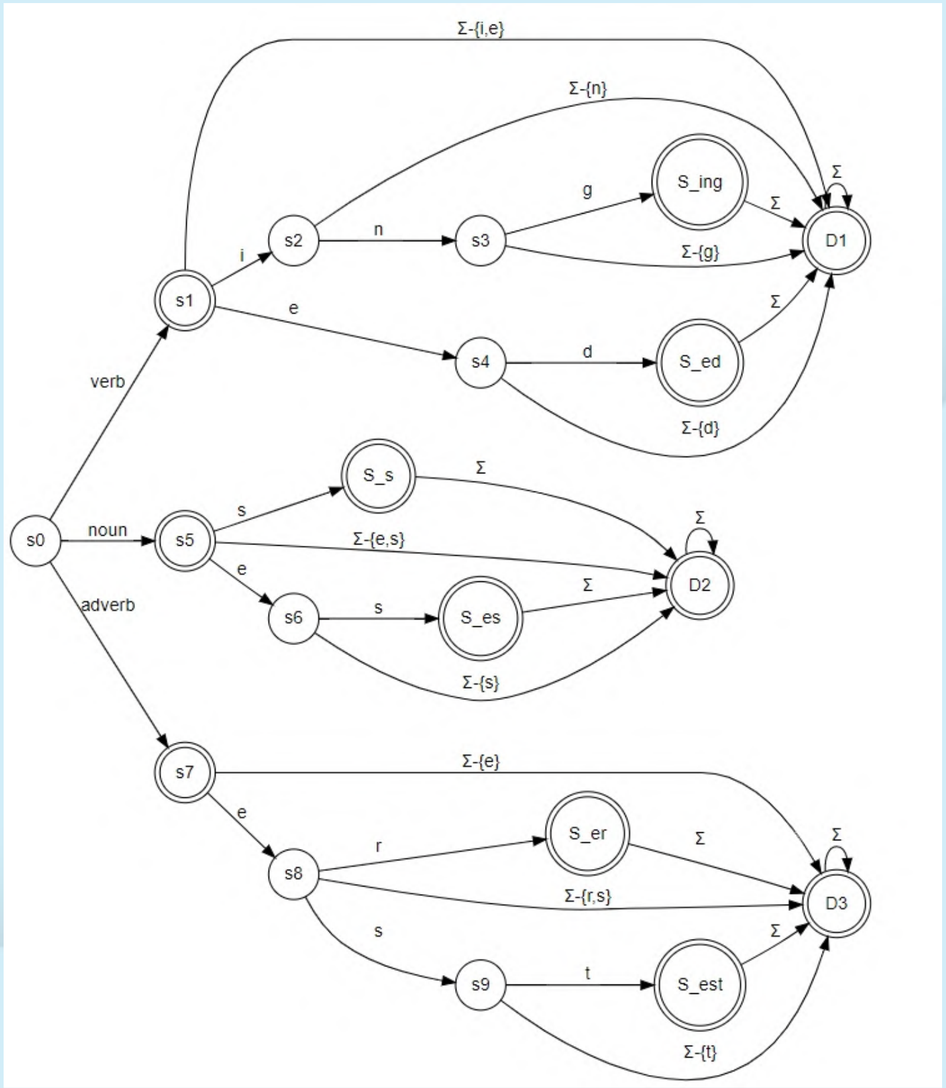
Transition Table For Generalised NFA

S_0	Verb	S_v
S_0	Noun	S_n
S_0	Adverb	S_adv
S_v	ing	S_ing
S_v	ed	S_ed
S_n	s	S_s
S_n	es	S_es
S_adv	er	S_er
S_adv	est	S_est

Deterministic finite automata

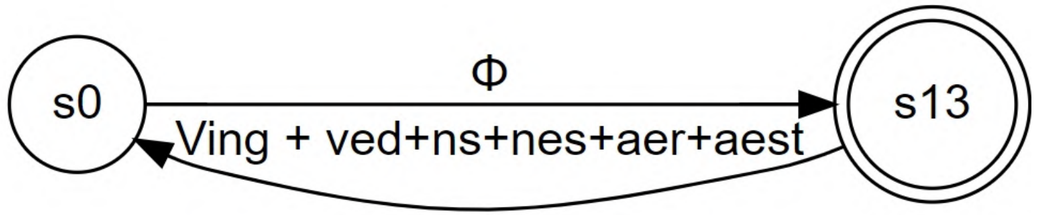


DFA Generalised Diagram

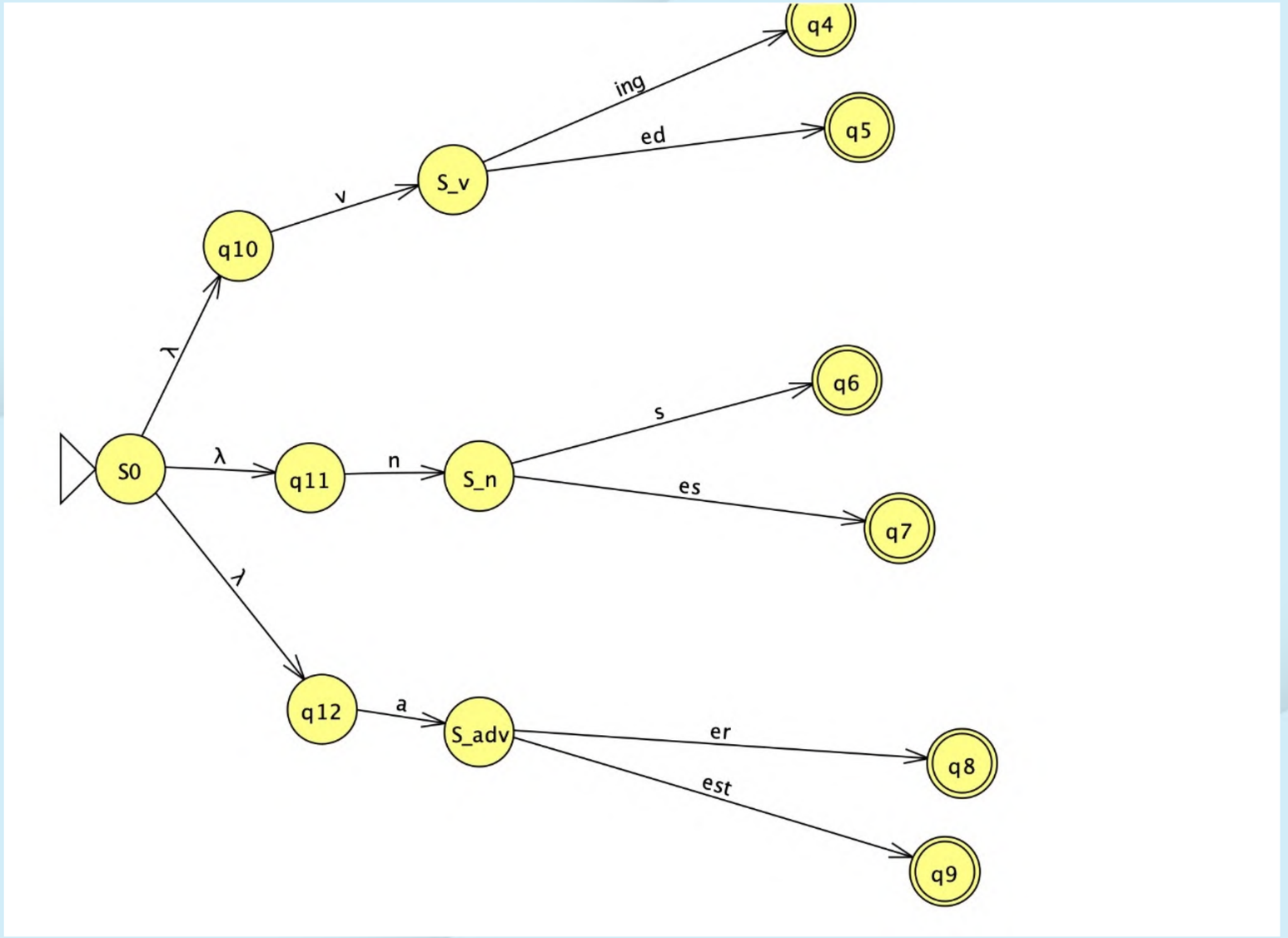


Generalized

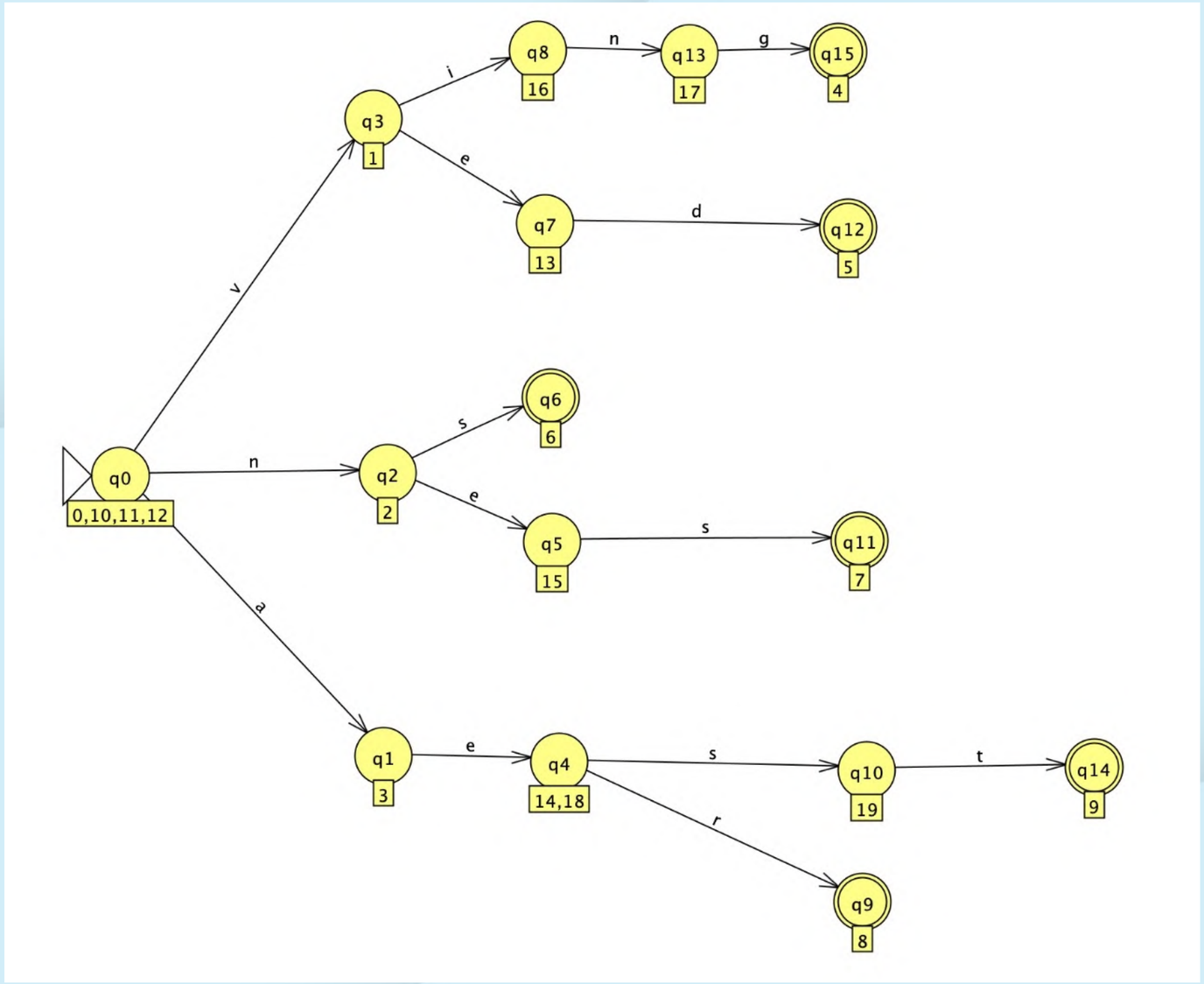
Regular Expressions



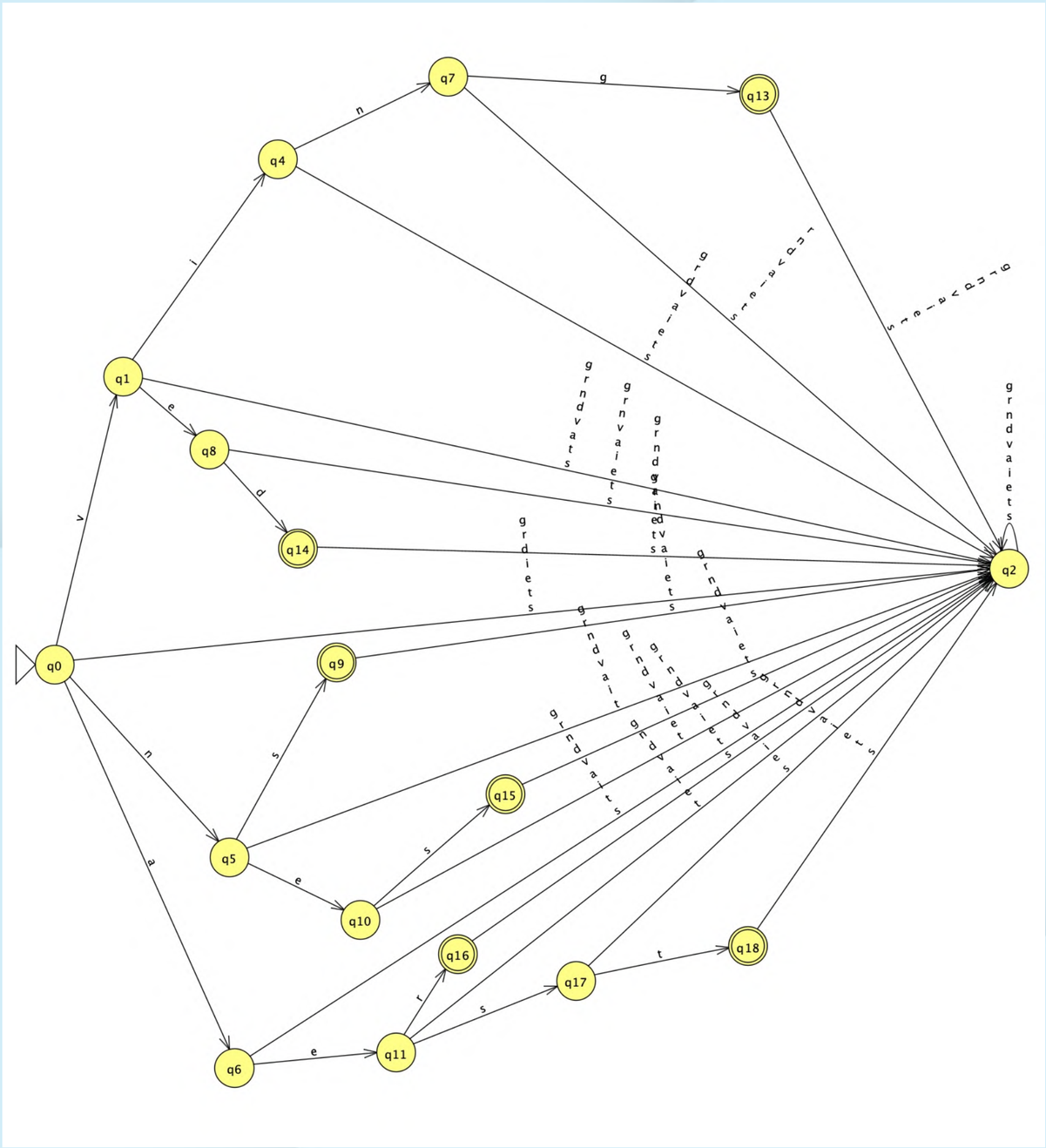
BUILDING ε-NFA USING JFLAP



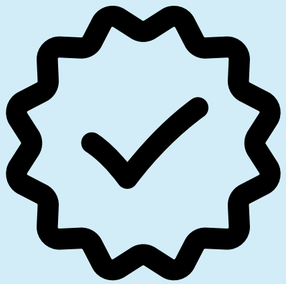
CONVERTING ε -NFA TO DFA



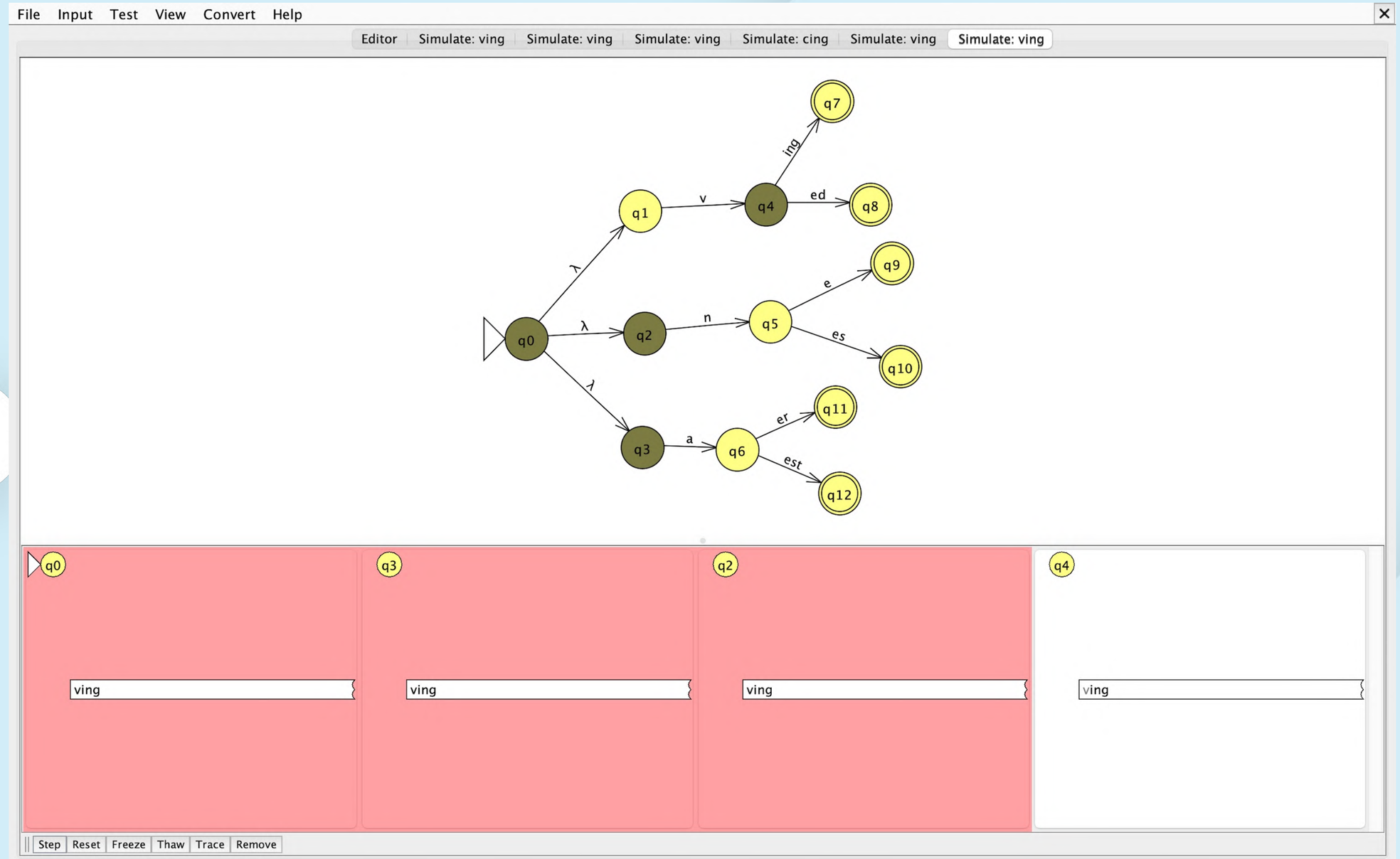
MINIMIZATION OF DFA



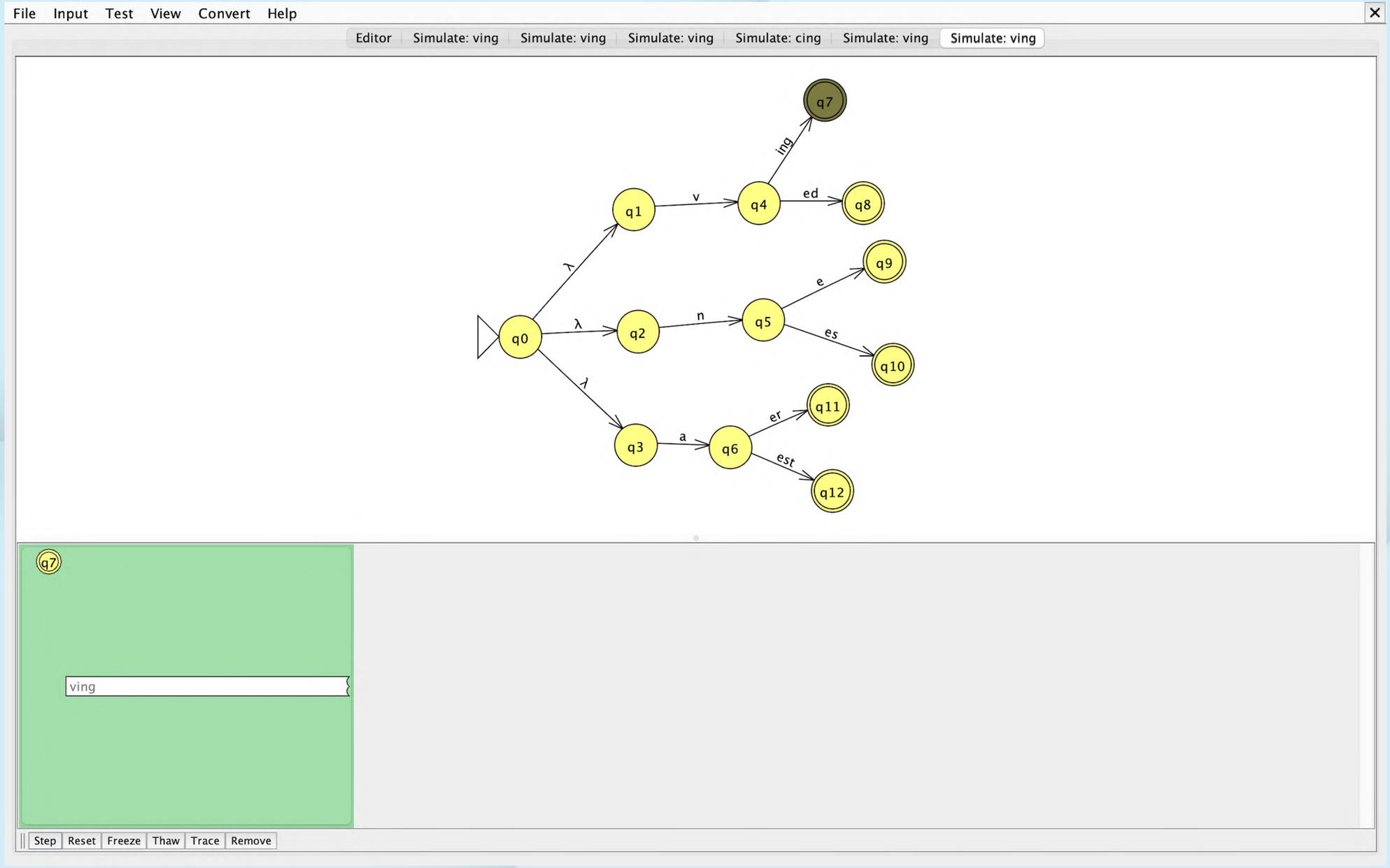
VERIFICATION USING JFLAP



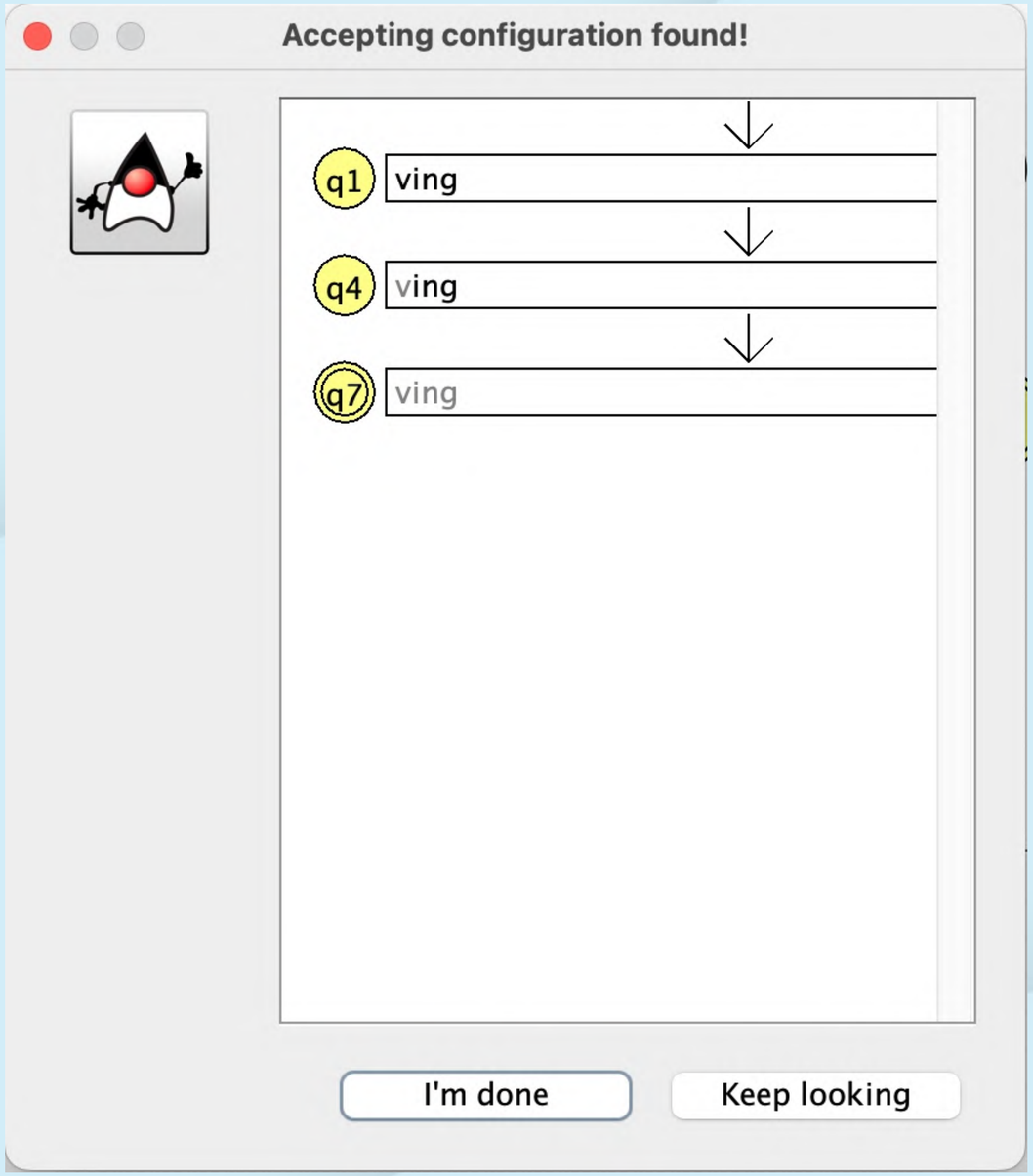
VERIFICATION OF STRING



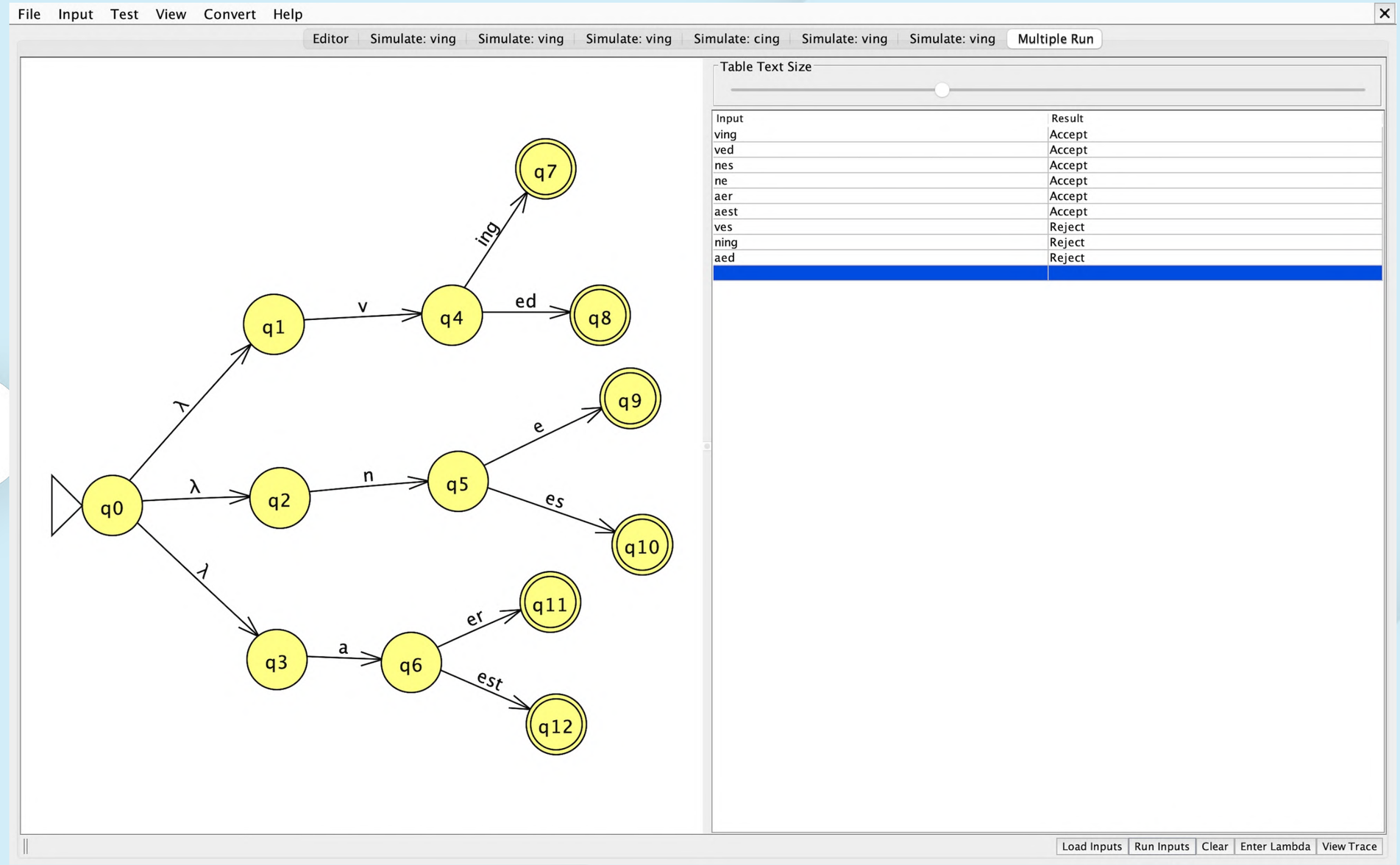
VERIFICATION OF STRING



STRING ACCEPTANCE



CHECKING MULTIPLE STRINGS



FST

Inflection

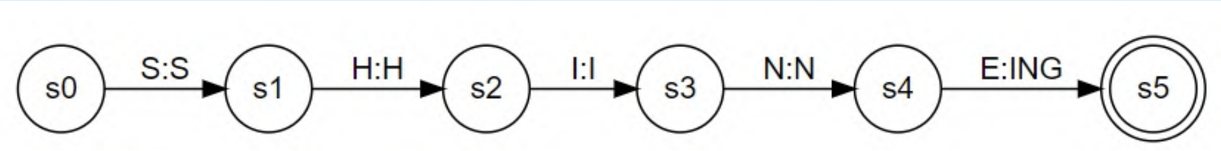
PRODUCTION RULES

Rule 1

Dropping the final -e

Drop –e if and only if it is preceded by a consonant or by the letter –u

Examples:
Shine → Shining
write → writing



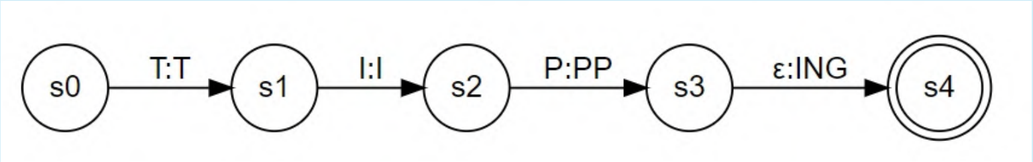
PRODUCTION RULES

Rule 2

Double Constant

Double a final single -n, -p, -t, -r if and only if it is preceded by a single vowel.

Examples:
Stop → Stopping
Tip → Tipping



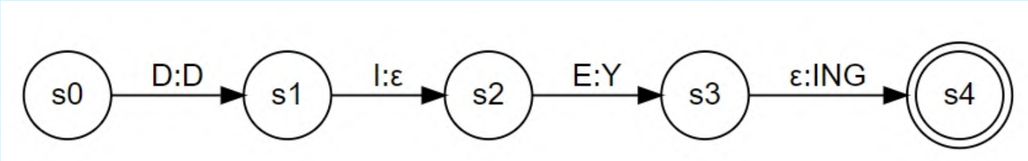
Exceptions : verbs ending with –er and –en (Gather ➔ Gathering || Happen ➔ Happening)

PRODUCTION RULES

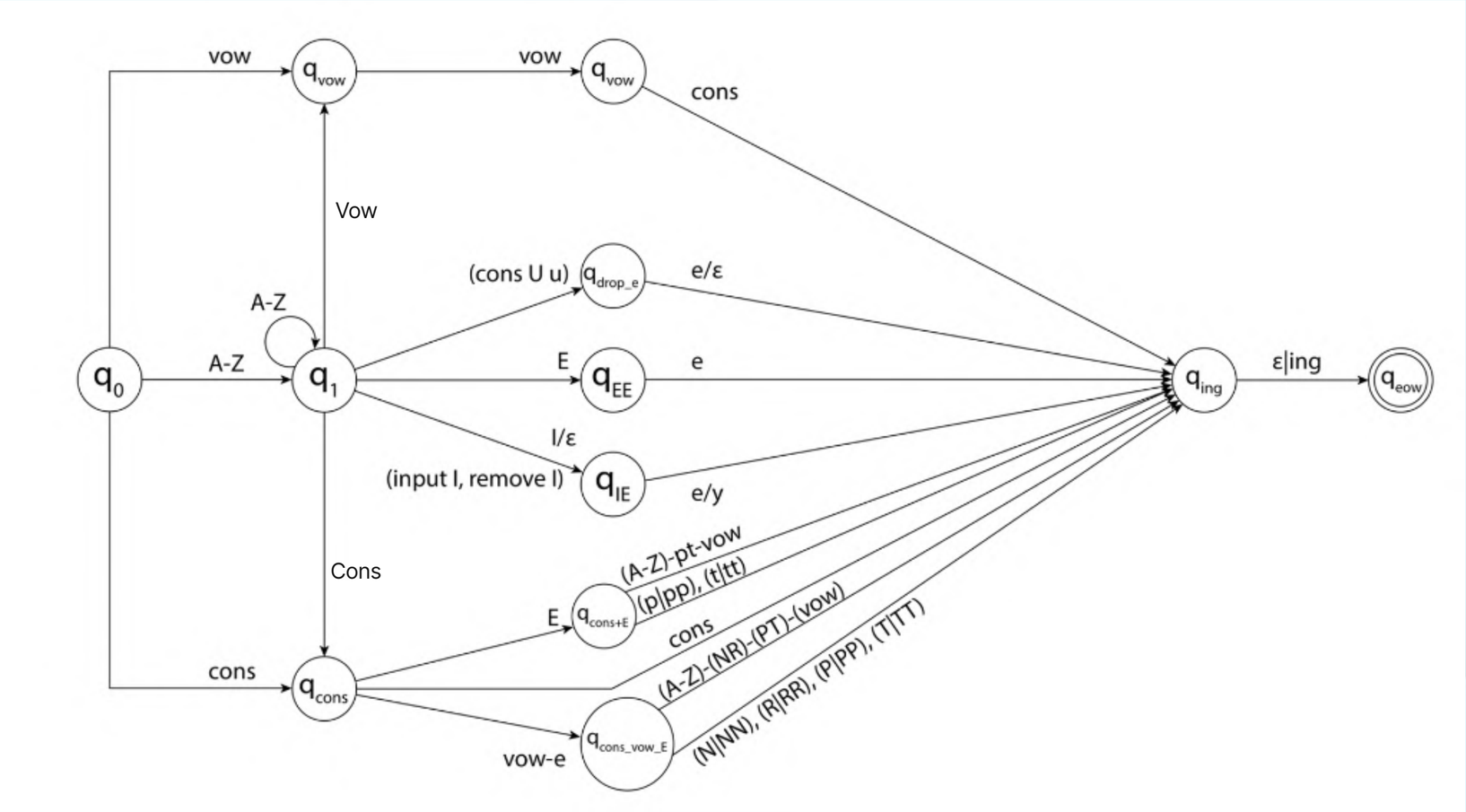
Rule 3

Change final -ie to –y
Example : die , tie

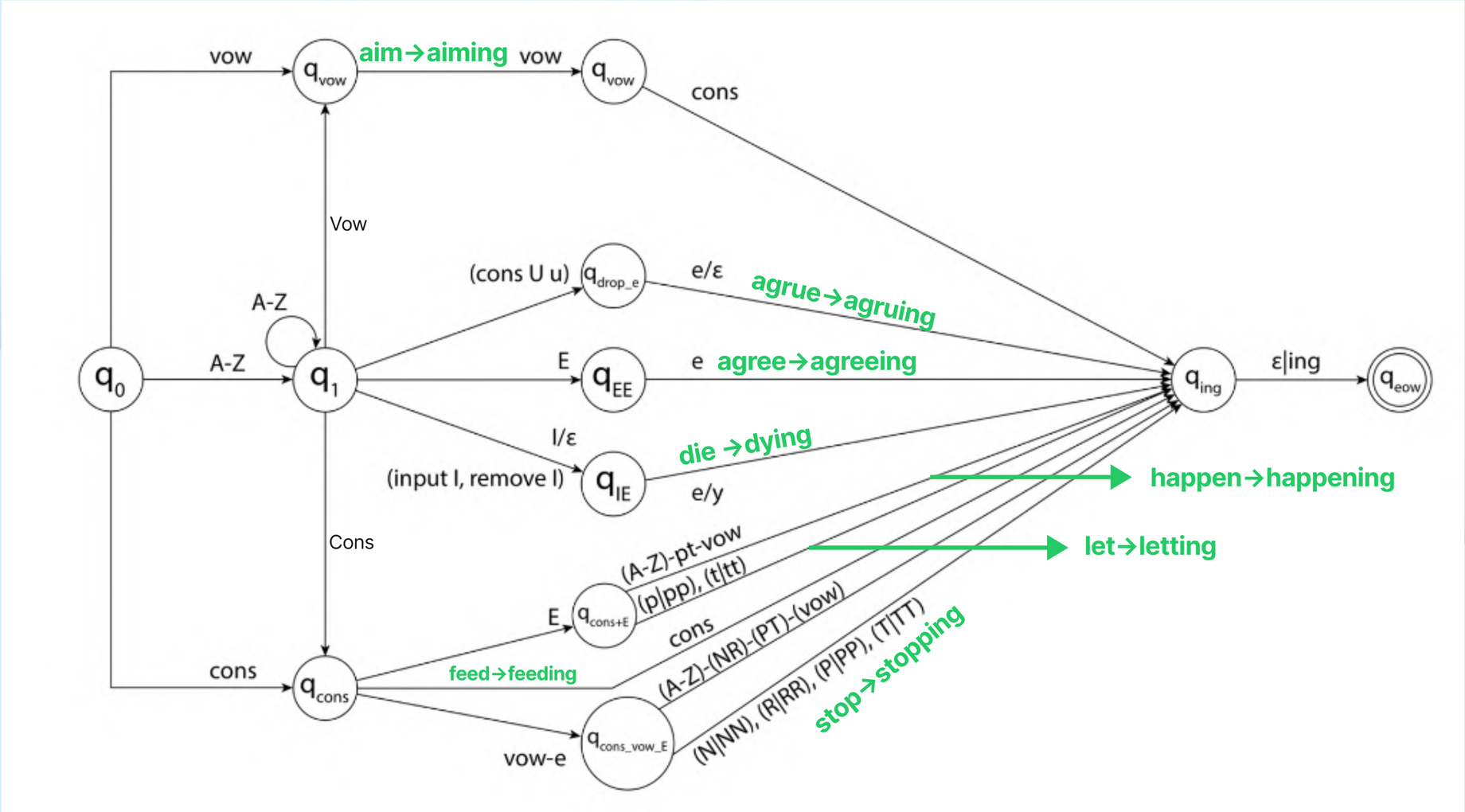
Examples:
die→ dying



FST


vow
vow
cons
Vow
(cons U u)
e/ε
A-Z
A-Z
E
e
ε|ing
l/ε
(input l, remove l)
e/y
Cons
(A-Z)-pt-vow
(p|pp), (t|tt)
E
cons
(A-Z)-(NR)-(PT)-(vow)
cons
(N|NN), (R|RR), (P|PP), (T|TT)
vow-e

FST


vow
aim→aiming
vow
cons
Vow
A-Z
(cons U u)
e/ε
agrue→agruing
A-Z
E
e
agree→agreeing
ε|ing
l/ε
die →dying
(input I, remove l)
e/y
happen→happening
Cons
(A-Z)-pt-vow
(p|pp), (t|tt)
let→letting
E
cons
(A-Z)-(NR)-(PT)-(vow)
cons
feed→feeding
stop→stopping
(N|NN), (R|RR), (P|PP), (T|TT)
vow-e

CODE

```python
class Transition:
    """The Basic Transition implementation of a state is defined"""

    def __init__(self, InState, InString, OutState, OutString = None):
        """ Defines the basic transition properties and intialized with Argument
            Instate    -   Current state ID
            Instring   -   Transition string input
            OutString  -   Transition output sting
            OutState   -   Next state's ID of the transition
        """

        self.InState = InState
        self.InString = InString
        self.OutString = OutString
        self.OutState = OutState

    def Output(self):
        return [self.OutString, self.OutState]
```

```python
class State:
    """ This is a class to define the properties of an individual State in a automata"""

    def __init__(self, name, FinalState):
        """ Initailize the State with Arguement:
            ID           -  Unique Identifier for the State (String or Integer)
            StateType    -  Defines the if the State is Terminal State or vice versa (Boolean)
            Transition   -  COntains all transitions from the State (Dictionary)"""

        self.ID = name
        self.StateType = FinalState
        self.Transition = dict()

    def addTransition(self, InString, OutString, OutState):
        """ Appends a Transition into the class variable 'Transition'
            Arguement of the method:
            Instring    -  Transistion string input
            Outstring   -  Output string of the transition
            Outstate    -  Next state of the transition

            Return:
            None
            """

        if InString in self.Transition.keys():
            #print("Transition with same Instring available")
            self.Transition[InString].append(Transition(self, InString,OutString= OutString, OutState = OutState))
        else:
            #print("New input string Transition")
            self.Transition[InString] = []
            self.Transition[InString].append(Transition(self, InString, OutString = OutString, OutState = OutState))
```

```python
def CheckTransition(self, InputAlphabet):

    if InputAlphabet not in self.Transition.keys():
        #print("No Transition as such")
        return [-1]
    elif InputAlphabet in self.Transition.keys():

        Transition_Output_List = []
        for Transit in self.Transition[InputAlphabet]:
            Transition_Output_List.append(Transit.Output())

        if '' in self.Transition.keys() and '' != InputAlphabet:
            for epilsonTransit in self.Transition['']:
                Transition_Output_List.append(epilsonTransit.Output())

        return Transition_Output_List
```

```python
class Automata:
    """ Basic class to define the automata's structure and its transitions. Further this automata model and be deployed in acceptor or

    def __init__(self, StateCount, Alphabet, OutAlphabets, InitialState, FinalStaeCount):

        self.StatesCount = StateCount
        self.Alphabet = Alphabet
        self.OutAlphabets = OutAlphabets
        self.InitialState = InitialState
        self.FinalStaeCount = FinalStaeCount

        self.States = dict()

    # Methods for building the automata structure

    def AddState(self, ID, Type):

        if self.StatesCount >= len(self.States):
            self.States[ID] = State(ID, Type)
        else:
            print("Exceeding the number of states!!!")

    def AddStateTransition(self, InState, InString, OutString, OutState):

        if InState in self.States.keys() and OutState in self.States.keys():
            In = self.States[InState]
            In.addTransition(InString, OutString, OutState)
            return
        else:
            return -1

    # Create a new Transition object for a state

    def AddsetTransition(self, InState, set ,OutState):

        for a in set:
            PCode = self.AddStateTransition(InState, a, a, OutState)
            if PCode == -1:
                print(f'No state : {InState} present for the Transition |OR| No state : {OutState} present for the Transition')
                break
```

```python
def AddsetEpilsonTransition(self, InState, set ,OutState):

    for a in set:
        PCode = self.AddStateTransition(InState, a, '', OutState)
        if PCode == -1:
            print(f'No state : {InState} present for the Transition |OR| No state : {OutState} present for the Transition')
            break


def AddEpilsonTransition(self, InState, OutState):

    PCode = self.AddStateTransition(InState, '', '', OutState)
    if PCode == -1:
        print(f'No state : {InState} present for the Transition |OR| No state : {OutState} present for the Transition')

def AddEpilsonStringTransition(self, InState, OutString, OutState):

    PCode = self.AddStateTransition(InState, '', OutString, OutState)
    if PCode == -1:
        print(f'No state : {InState} present for the Transition |OR| No state : {OutState} present for the Transition')
```

```python
class Machine:

    def __init__(self, Automata, InputString):
        self.Automata = Automata
        self.IntialState = Automata.InitialState
        self.InputString = InputString

    def Run(self,CurrentState, I = None, String = None):
        #print("Current State: ",CurrentState, " Current Input: ", I)

        if I is None:
            #print("checking for terminal case --->")
            Terminal_Transition_outputs = self.Automata.States[CurrentState].CheckTransition(InputAlphabet = '')
            if len(Terminal_Transition_outputs) > 0 and Terminal_Transition_outputs[0] != -1:
                Terminal_Transition_outputs = self.Automata.States[CurrentState].CheckTransition(InputAlphabet = '')

                Output = []
                for Transit in Terminal_Transition_outputs:
                    if Transit == -1:
                        continue
                    Sub_Output = self.Run(Transit[1])
                    #print(Sub_Output)
                    for S in Sub_Output:

                        S.insert(0,Transit[0])
                        Output.append(S)
                return Output


            elif self.Automata.States[CurrentState].StateType == True:
                #print([[True]])
                return [[True]]
            else:
                #print([[False]])
                return [[False]]
```

```
        else:
            Output = []
            Transition_Outputs = self.Automata.States[CurrentState].CheckTransition(InputAlphabet = I)

            if Transition_Outputs[0] == -1:
                return [[False]]

            else:

                for Transit_Output in Transition_Outputs:

                    if Transit_Output == -1:
                        continue

                    elif len(String) == 0:
                        Sub_Transition_Output = self.Run(Transit_Output[1], I = None, String= None)
                    else:
                        Sub_Transition_Output = self.Run(Transit_Output[1], I = String[0], String= String[1:])

                    for S in Sub_Transition_Output:
                        S.insert(0,Transit_Output[0])
                        Output.append(S)
                return Output

    def Machine_FST_Output(self):

        print("Initiating machine")
        OutputList = self.Run(self.IntialState, I= self.InputString[0], String=self.InputString[1:])
        print("Completing the Generation...")
        for Output in OutputList:
            if True in Output:
                Output.remove(True)
                Generated_String = "".join(Output)
                print("Input String: " + self.InputString)
                print("Generated Output String -->" + Generated_String)
            else:
                print("Input String not Accepted!!!")
```

BUILDING FST

```
A = Automata(StateCount=12, Alphabet=set('abcdefghijklmnopqrstuvwxyz'), OutAlphabets= None, InitialState='q0', FinalStaeCount=1)

#Adding States
A.AddState(ID='q0',Type=False)
A.AddState(ID='q1',Type=False)
A.AddState(ID='q_ing',Type=False)
A.AddState(ID='q_drop_E',Type=False)
A.AddState(ID='q_EE',Type=False)
A.AddState(ID='q_IE',Type=False)
A.AddState(ID='q_VOWS',Type=False)
A.AddState(ID='q_VOWS2',Type=False)
A.AddState(ID='q_CONS',Type=False)
A.AddState(ID='q_CONS+E',Type=False)
A.AddState(ID='q_CONS_VOWS-E',Type=False)
A.AddState(ID='q_EOW',Type=True)

A2Z=set('abcdefghijklmnopqrstuvwxyz')
VOWS=set('aieou')
CONS=A2Z-VOWS
U=set('u')
E=set('e')
I=set('i')
PT=set('pt')
NR=set('nr')
```

Transitions

```
#Adding Transitions
#AddsetTransition(self, InState, set ,OutState)
A.AddsetTransition('q0', VOWS , 'q_VOWS')
A.AddsetTransition('q_VOWS', VOWS, 'q_VOWS2')
A.AddsetTransition('q_VOWS2',CONS,'q_ing')
#AddEpilsonStringTransition(self, InState, OutString, OutState)
A.AddEpilsonStringTransition('q_ing','ing','q_EOW')
```

```
##check
#AddsetTransition(self, InState, set ,OutState)
A.AddsetTransition('q0', A2Z ,'q1')
A.AddsetTransition('q1', A2Z ,'q1')
A.AddsetTransition('q1', CONS.union(U) ,'q_drop_E')
#AddsetEpilsonTransition(self, InState, set ,OutState)
A.AddsetEpilsonTransition('q_drop_E',E,'q_ing')
A.AddsetTransition('q1', E ,'q_EE')
A.AddsetTransition('q_EE', E ,'q_ing') ##########################
A.AddsetEpilsonTransition('q1',I,'q_IE')
#AddStateTransition(self, InState, InString, OutString, OutState)
A.AddStateTransition('q_IE', 'e','y','q_ing')
A.AddsetTransition('q1',CONS,'q_CONS')
A.AddsetTransition('q1',VOWS,'q_VOWS')
```

```
#AddsetTransition(self, InState, set ,OutState)
A.AddsetTransition('q0', CONS ,'q_CONS')
A.AddsetTransition('q_CONS', E ,'q_CONS+E')
A.AddsetTransition('q_CONS+E',A2Z-PT-VOWS,'q_ing')
#AddStateTransition(self, InState, InString, OutString, OutState
A.AddStateTransition('q_cons+E','p','pp','q_ing')
A.AddStateTransition('q_cons+E','t','tt','q_ing')
A.AddsetTransition('q_CONS', VOWS-E ,'q_CONS_VOWS-E')
A.AddsetTransition('q_CONS_VOWS-E',A2Z-NR-PT-VOWS,'q_ing')
A.AddStateTransition('q_CONS_VOWS-E','p','pp','q_ing')
A.AddStateTransition('q_CONS_VOWS-E','n','nn','q_ing')
A.AddStateTransition('q_CONS_VOWS-E','r','rr','q_ing')
A.AddStateTransition('q_CONS_VOWS-E','t','tt','q_ing')
```

OUTPUT

TESTING WITH INPUTS

```
O = Machine(A,"bite")
A.InitialState
O.Run(A.InitialState, I=O.InputString[0],String=O.InputString[1:])
```

```
[['b', 'i', 't', 'e', False],
 ['b', 'i', 't', 'e', False],
 ['b', 'i', 't', 'e', False],
 ['b', 'i', 't', '', 'ing', True],
 ['b', 'i', 't', 'e', False],
 ['b', '', False],
 ['b', 'i', False],
 ['b', 'i', 'tt', False]]
```

```
O.Machine_Output()
```

```
Initiating machine
Completing the Generation...
Input String: bite
Generated Output String -->biting
```

DEMO

Thank You