The purpose of this assignment is to write small finite state morphologies and phonologies. Work in Python, using the definition methodology. See the notebook for Russian for the pattern of solution.

Problems 1 and 2 list forms from the languages Lamba and Tagalog. In each problem, construct in a finite state lexicon that has the following form. Use these exact names for the transducers you define. Submit a notebook and a pdf of a run of the notebook.

In [ ]:

```python
import hfst_dev as hfst
import graphviz
import random
```

In [ ]:

```python
import random

def sample_input(x,n=5,cycles=3):
        x2 = x.copy()
        x2.input_project()
        x2.minimize()
        return(random.sample(set(x2.extract_paths(max_cycles=3).keys()),n))
def sample_output(x,n=5,cycles=3):
        x2 = x.copy()
        x2.output_project()
        x2.minimize()
        return(random.sample(set(x2.extract_paths(max_cycles=3).keys()),n))
```

In [ ]:

```python
def apply_rules(u,rs):
    m = hfst.regex(" ".join([x for x in u]))
    print(list(m.extract_paths(max_cycles=3).keys())[0])
    for r in rs:
        m.compose(r)
        m.output_project()
        m.minimize()
        print(list(m.extract_paths(max_cycles=3).keys())[0])
```

In [ ]:

```python
def parse(u,Lexicon):
    um = hfst.regex(" ".join([x for x in u]))
    inv = Lexicon.copy()
    inv.invert()
    um.compose(inv)
    um.output_project()
    um.minimize()
    return(list(um.extract_paths(max_cycles=3).keys()))
```

1 FOR LAMBA

(i) Morpheme relation

A relation LambaM or TagalogM that has a multi-character symbol corresponding to an English gloss on the upper side, and an underlying spelling for the language on the lower side. For instance suppose that in your analysis the underlying spelling of WATER is {aa}. Then you should see behavior like this in foma.

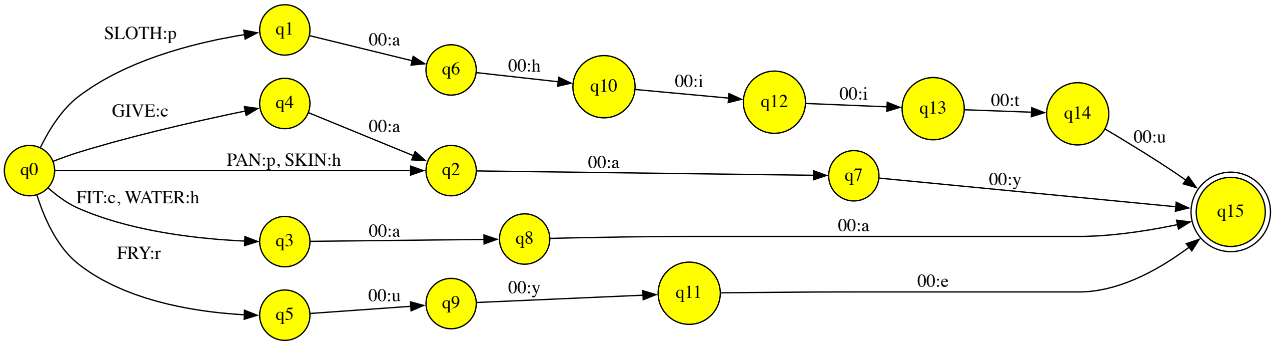
set print-space ON; regex [WATER .o. LambaM].l print words a a

The symbol WATER has been mapped the to the underlying spelling {aa}.

In [ ]:

```
Lamba_expr_N = ''' [WATER .x. {haa}]  |
[SLOTH .x. {pahiitu}] |
[PAN .x. {pay}] |
[SKIN .x. {hay}]'''

Lamba_expr_V = '''
[FIT .x. {caa}] |
[GIVE .x. {caay}] |
[FRY .x. {ruye}]
'''

LambaN = hfst.regex(Lamba_expr_N)
LambaV = hfst.regex(Lamba_expr_V)
LambaM = hfst.regex(Lamba_expr_N + ' | ' + Lamba_expr_V)
LambaM.view()
```

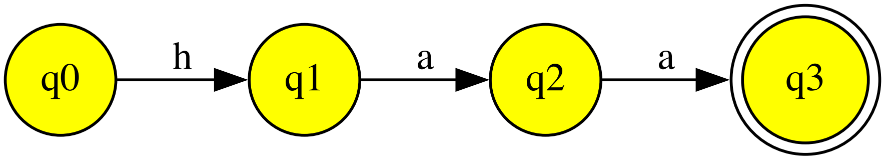
Out[ ]:



In [ ]:

```
defs = {'LambaN':LambaN , 'LambaV':LambaV, 'LambaM':LambaM}
Lamba1 = hfst.regex('[WATER .o. LambaM].l', definitions=defs)
Lamba1.view()
```

Out[ ]:



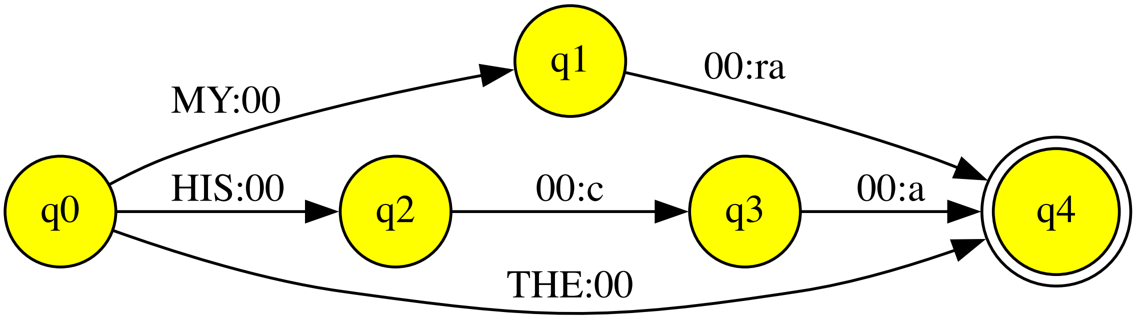
(ii) A set LambaPHRASE or TagalogPHRASE of underling morpheme sequences for the examples in the table. Each element is a sequence abstract morphemes. You need to figure out the optimal order, and define the phrases using a Foma definition or sequence of definitions. For instance if '1STGEN' and 'PAN' are morphemes, the underlying form of 'capay' in could be '1STGEN PAN' or 'PAN 1STGEN', and this should be an element of LambaPHRASE. (1STGEN is supposed to suggest first person genitive.)

Nouns

comment: i would assume here that noun is equal to the noun

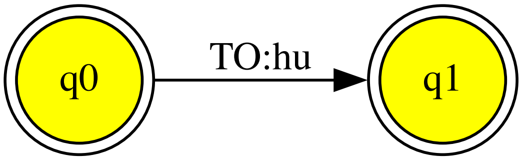
In [ ]:
```
Left_str_L = """
[[HIS:0] 0:c 0:a ] |
[[THE:0]] |
[[MY:0] 0:ra ]
"""
Left_L = hfst.regex(Left_str_L)
Left_L.view()
```

Out[ ]:



In [ ]:
```
Right_str_L = """
[ TO: hu ] |
[ 0:0 ]
"""
Right_L = hfst.regex(Right_str_L)
Right_L.view()
```

Out[ ]:



Verbs

comment: i'm constructing the nouns and the verbs in different way and take the union of two languages. (disjunct)

In [ ]:

```
Left_str_V = """
[[HE:0] 0:ca ] |
[[I:0] 0:ra ]
"""
Left_V = hfst.regex(Left_str_V)

Right_str_V = """
[0:0] |
[[IT:0] 0:ra ]
"""
Right_V = hfst.regex(Right_str_V)
```

In [ ]:

```
LambaPHRASE = Left_L.copy()
LambaPHRASE.input_project()

separator = hfst.regex('" ":0')
separator_up = separator.copy()
separator_up.input_project()

LambaN_up = LambaN.copy()
LambaN_up.input_project()

right_up = Right_L.copy()
right_up.input_project()

LambaPHRASE.concatenate(separator_up)
LambaPHRASE.concatenate(LambaN_up)
LambaPHRASE.concatenate(separator_up)
LambaPHRASE.concatenate(right_up)

#  verbs
V_PHRASE = Left_V.copy()
V_PHRASE.input_project()

V_PHRASE.concatenate(separator_up)
V_PHRASE.concatenate(LambaV)
V_PHRASE.concatenate(separator_up)

right_up_V = Right_V.copy()
V_PHRASE.concatenate(right_up_V)

V1 = V_PHRASE.copy()
LambaPHRASE.disjunct(V1)
```

In [ ]:

```
sample_input(LambaPHRASE)
```

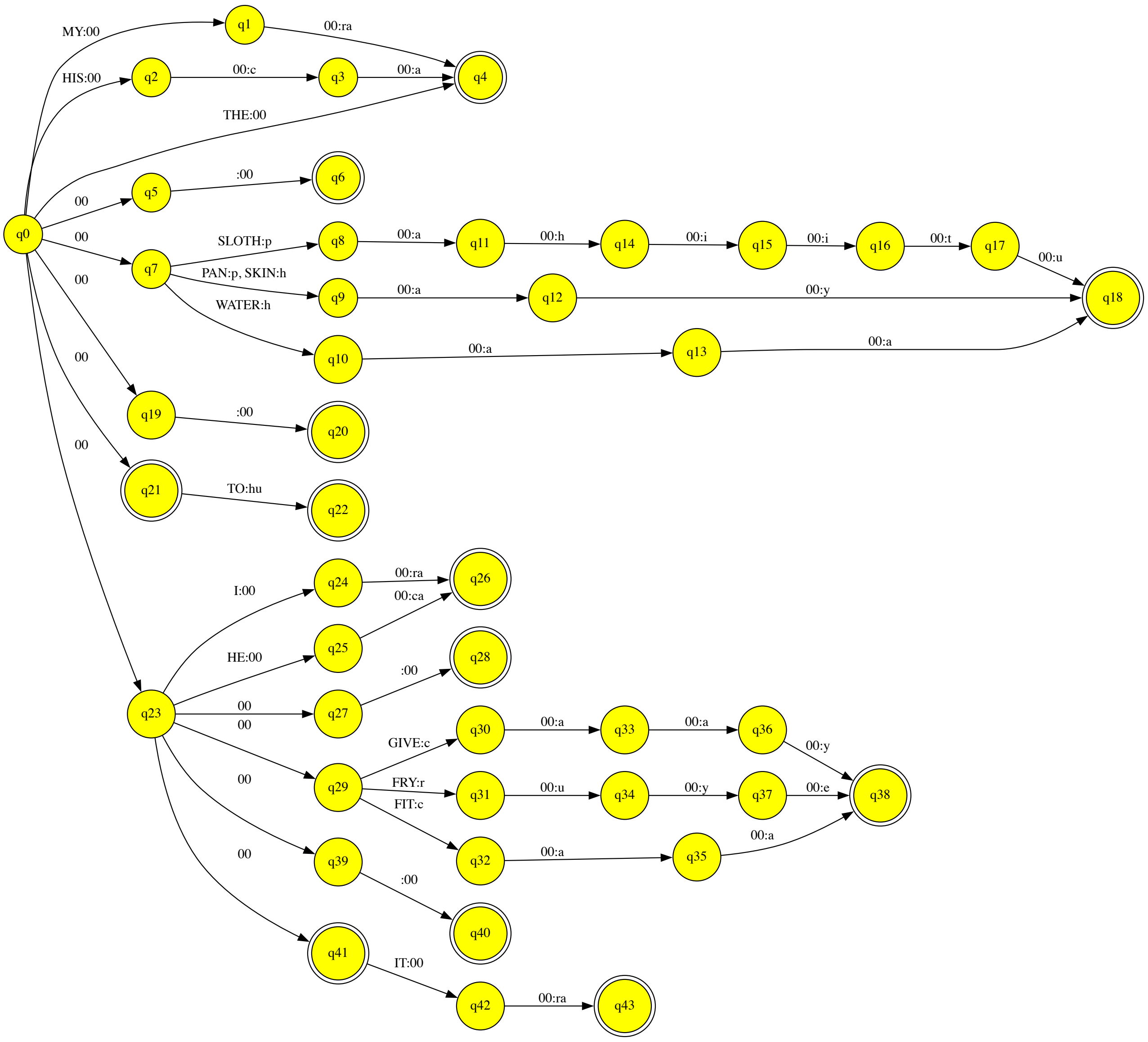
Out[ ]:

```
['HIS WATER ', 'THE SKIN TO', 'HIS WATER TO', 'I GIVE ', 'MY PAN TO']
```

In [ ]:

```
M = Left_L.copy()
M.disjunct(separator)
M.disjunct(LambaN)
M.disjunct(separator)
M.disjunct(Right_L)

M2 = Left_V.copy()
M2.disjunct(separator)
M2.disjunct(LambaV)
M2.disjunct(separator)
M2.disjunct(Right_V)

M.disjunct(M2)

M.view()
```

Out[ ]:



In [ ]:

```
LambaMOR = M.copy()
LambaMOR.repeat_plus()
```

In [ ]:

```
x = LambaPHRASE.copy()
x.compose(LambaMOR)
# sample_input(x)
sample_output(x)
```

Out[ ]:

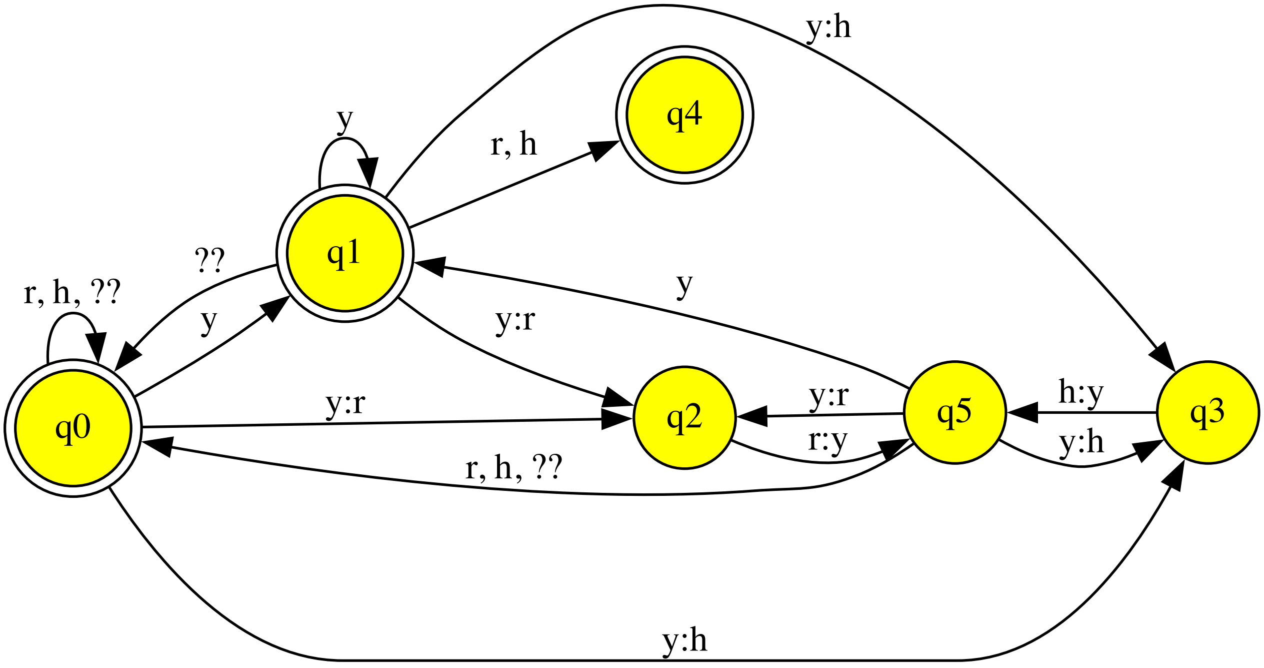
```
['haa', 'cahay', 'cahaahu', 'capayhu', 'hayhu']
```

Rules 1 y-metathesis

In [ ]:

```
MT = hfst.regex(" [y h] -> [ h y], [y r] -> [r y] ||  _ ?")
MT.view()
```

Out[ ]:



In [ ]:

```
apply_rules('payhu',[MT])
apply_rules('cacaayra',[MT])
```

```
payhu
pahyu
cacaayra
cacaarya
```

rule2 y insertion

```
In [ ]: IY = hfst.regex(" h -> hy, p ->[p l y], c->cy, r->ry ||  [ r a ] _ [  a | u
        apply_rules('rahaa',[MT,IY])
        apply_rules('rapahiitu',[MT,IY])
        apply_rules('racaara',[MT,IY])
        apply_rules('raruye',[MT , IY])

        end_IY = hfst.regex(" [ r a ] -> [ r y a] ||  _ .#. ")

        apply_rules('raruyera',[end_IY, MT,IY])
```

```
rahaa
rahaa
rahyaa
rapahiitu
rapahiitu
raplyahiitu
racaara
racaara
racyaara
raruye
raruye
raryuye
raruyera
raruyerya
raruyerya
raryuyerya
```

rule3 ay-e vowel fronting

```
In [ ]: EY = hfst.regex(" [a a y] -> e e, [a y] -> e || [ r a ? ] _  ")
        apply_rules('rahay',[EY, MT, IY])
        apply_rules('racaay',[EY ,MT, IY])
        apply_rules('racaayra',[end_IY, EY ,MT, IY])
```

```
rahay
rahe
rahe
rahe
racaay
racee
racee
racee
racaayra
racaayrya
raceerya
raceerya
raceerya
```

In [ ]:

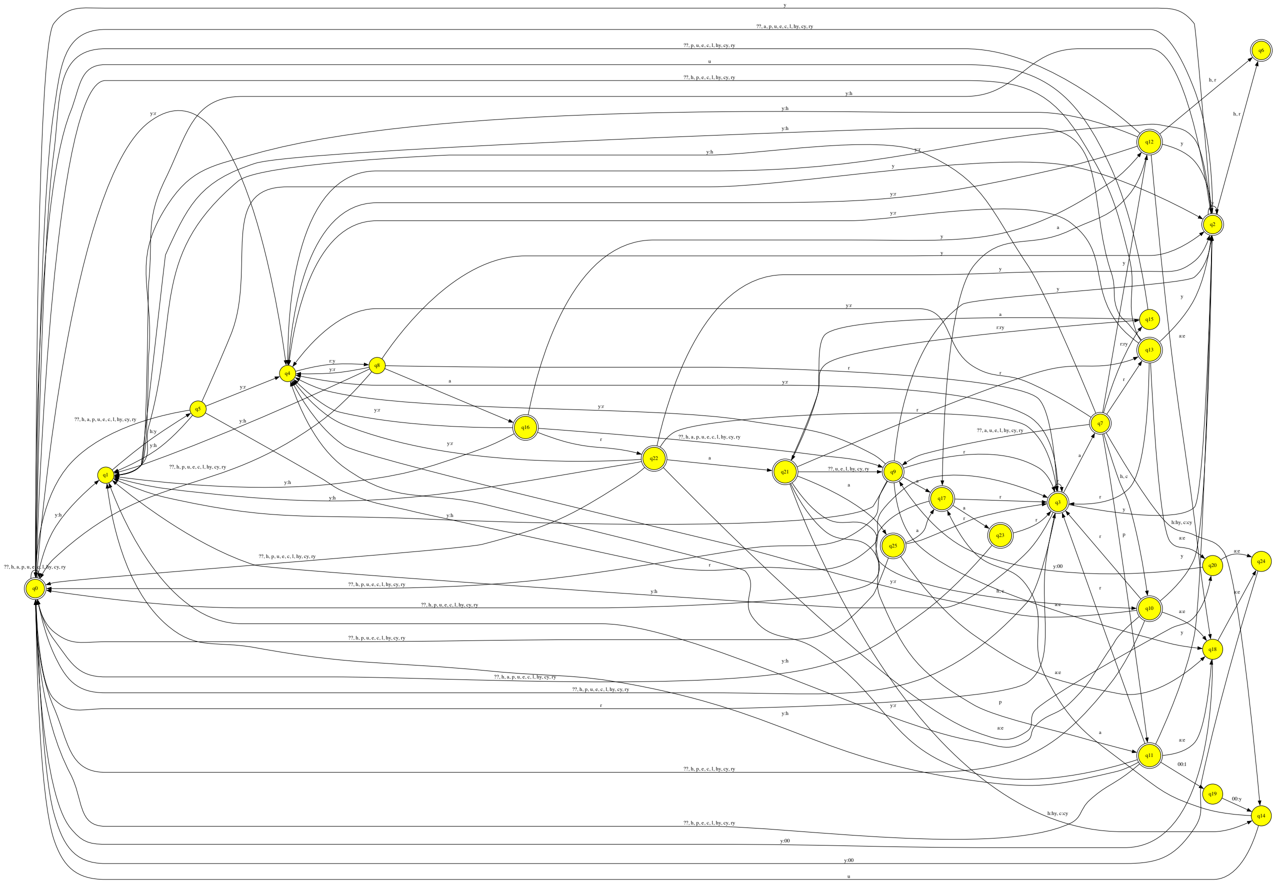
```
defs['MT'] = MT
defs['IY'] = IY
defs['EY'] = EY
defs['endIY'] = end_IY
LambaPHON = hfst.regex(' endIY .o. EY .o.  MT  .o. IY ', definitions=defs)
LambaPHON.minimize()


apply_rules('rahay',[LambaPHON])
apply_rules('rahaahu',[LambaPHON])
apply_rules('racaay',[LambaPHON])
apply_rules('racaara',[LambaPHON])
```

```
rahay
rahe
rahaahu
rahyaahu
racaay
racee
racaara
racyaarya
```

In [ ]:

```
LambaPHON.view()
```

Out[ ]:



In [ ]:

```
defs['LambaPHON'] = LambaPHON
defs['LambaPHRASE'] = LambaPHRASE
defs['LambaMOR'] = LambaMOR
defs.keys()
```

Out[ ]:

```
dict_keys(['TagalogM', 'LambaPHON', 'LambaPHRASE', 'LambaMOR'])
```

In [ ]:

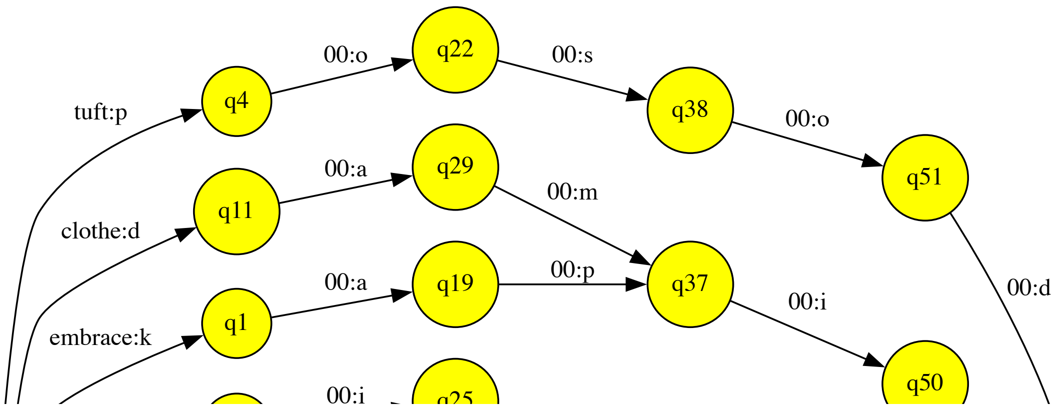
```
Lamba = hfst.regex('[LambaPHRASE .o. LambaMOR .o. LambaPHON]', definitions=d
Lamba.minimize()
```

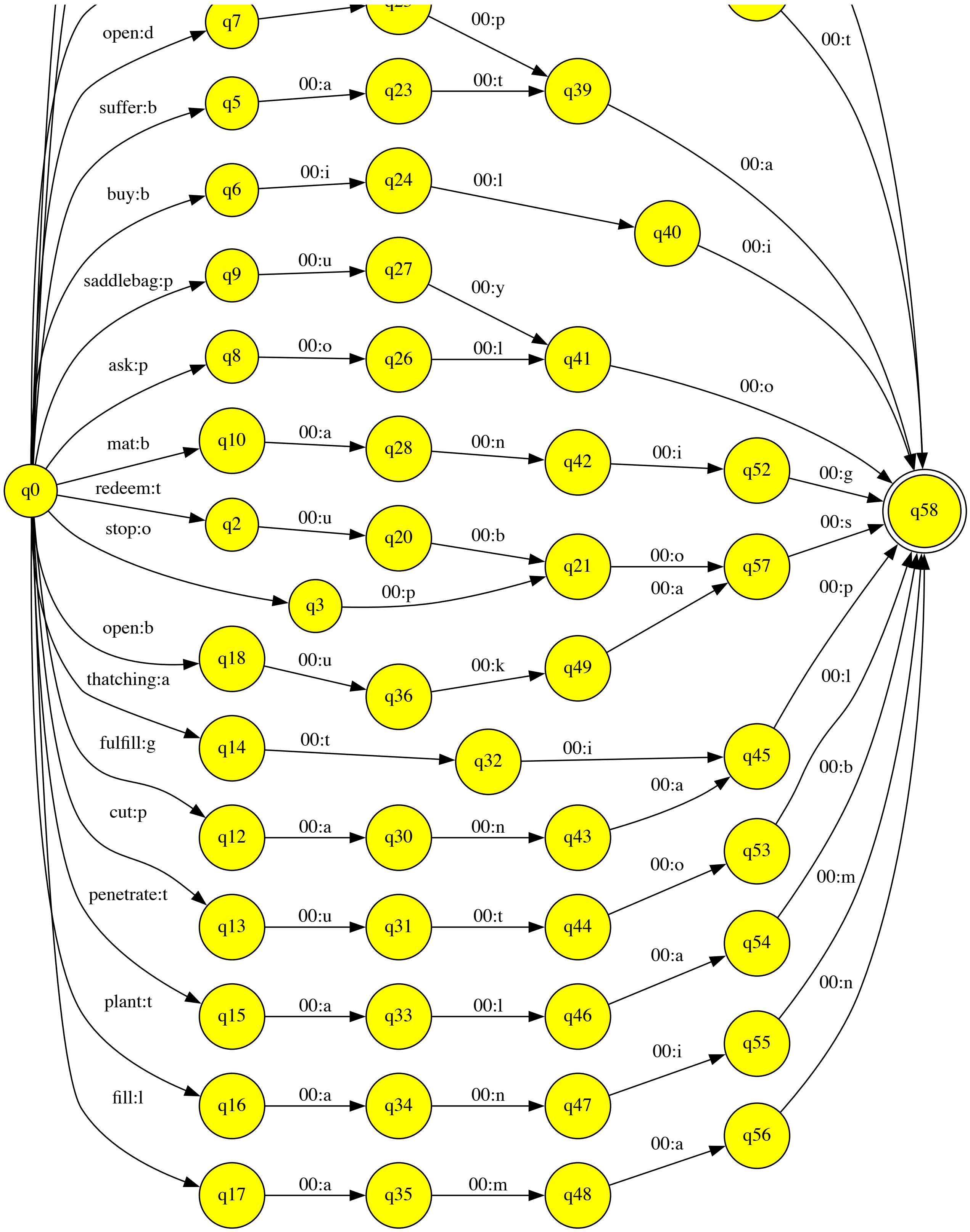
TAGALOG

In [ ]:

```
Tagalog_expr = ''' [open .x. {bukas}]  |
[embrace .x. {kapit}] |
[redeem .x. {tubos}] |
[stop .x. {opos}] |
[tuft .x. {posod}] |
[suffer .x. {bata}] |
[buy .x. {bili}] |
[open .x. {dipa}] |
[ask .x. {polo}] |
[saddlebag .x. {puyo}] |
[mat .x. {banig}] |
[clothe .x. {damit}] |
[fulfill .x. {ganap}] |
[cut .x. {putol}] |
[thatching .x. {atip}] |
[penetrate .x. {talab}] |
[plant .x. {tanim}]|
[fill .x. {laman}]
'''
TagalogM = hfst.regex(Tagalog_expr)
TagalogM.view()
```

Out[ ]:

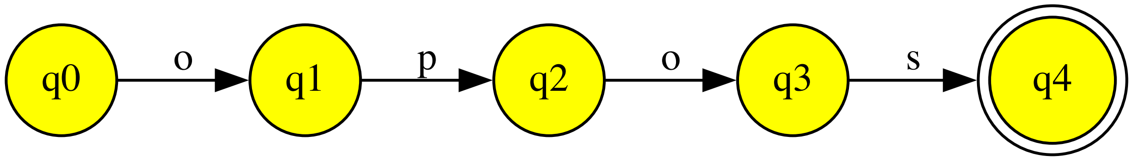




In [ ]:

```
defs = {'TagalogM':TagalogM}
Tagalog1 = hfst.regex('[stop .o. TagalogM].l', definitions=defs)
Tagalog1.view()
```

Out[ ]:



In [ ]:

```
infl = hfst.regex(' IN : in | AN : an | 0:0 ')
```

In [ ]:

```
TagalogPHRASE = TagalogM.copy()
TagalogPHRASE.input_project()
separator = hfst.regex('" ":0')
separator_up = separator.copy()
separator_up.input_project()
Infl_up = infl.copy()
Infl_up.input_project()
TagalogPHRASE.concatenate(separator_up)
TagalogPHRASE.concatenate(Infl_up)
```

In [ ]:

```
sample_output(TagalogPHRASE)
```

Out[ ]:

```
['saddlebag ', 'fill ', 'open IN', 'thatching IN', 'penetrate ']
```

In [ ]:

```
M = TagalogM.copy()
M.disjunct(separator)
M.disjunct(infl)
TagalogMOR = M.copy()
TagalogMOR.repeat_plus()
```

In [ ]:

```
x = TagalogPHRASE.copy()
x.compose(TagalogMOR)
sample_output(x)
```

Out[ ]:

```
['bili', 'biliin', 'tubosin', 'kapit', 'posodin']
```

In [ ]:

```
Cons = hfst.regex("b | d | k | g | l | m | n | p | r | s | t  | v | z | ŋ")
Stops = hfst.regex("b | d | k | g | p | t")
Vowel = hfst.regex("e | a | i | o | u")
defs = {"C":Cons,"V":Vowel, "S":Stops}
```

rule 1 Syncope

In [ ]:

```
Syncope =  hfst.regex(" V-> 0  || ?* V C _ C V n",definitions=defs)
apply_rules('bukasin',[Syncope])
apply_rules('bukasan',[Syncope])
apply_rules('kapatin',[Syncope])
apply_rules('tubosin',[Syncope])
```

```
bukasin
buksin
bukasan
buksan
kapatin
kaptin
tubosin
tubsin
```

Rule 2 O2U

In [ ]:

```
o2u = hfst.regex(" o -> u || ?* _ C o [C|0] V n",definitions=defs)
apply_rules('oposin',[o2u , Syncope])
apply_rules('posodin',[o2u, Syncope])
```

```
oposin
uposin
upsin
posodin
pusodin
pusdin
```

Rule 3 end vowel silence

In [ ]:

```
silence = hfst.regex(" V -> h || ?* _ V n",definitions=defs)
apply_rules('batain',[o2u , Syncope, silence])
apply_rules('bataan',[o2u, Syncope, silence])
apply_rules('biliin',[o2u, Syncope, silence])
apply_rules('dipaan',[o2u, Syncope, silence])
apply_rules('poloin',[o2u, Syncope, silence])
apply_rules('puyoin',[o2u, Syncope, silence])
```

```
batain
batain
batain
bathin
bataan
bataan
bataan
bathan
biliin
biliin
biliin
bilhin
dipaan
dipaan
dipaan
diphan
poloin
puloin
puloin
pulhin
puyoin
puyoin
puyoin
puyhin
```

the missing one first is: puyhan

```
In [ ]: apply_rules('puyoan',[o2u, Syncope, silence])
```

```
puyoan
puyoan
puyoan
puyhan
```

rule 4 nasal change

```
In [ ]: nasal_ng = hfst.regex(" n -> ŋ || ?* V _ i S V n",definitions=defs)
        apply_rules('banigin',[nasal_ng , o2u, Syncope, silence])
```

```
banigin
baŋigin
baŋigin
baŋgin
baŋgin
```

```
In [ ]: nasal_m = hfst.regex(" n -> m || ?* V _ a C V n",definitions=defs)
        apply_rules('ganapin',[nasal_ng , nasal_m, o2u, Syncope, silence])
```

```
ganapin
ganapin
gamapin
gamapin
gampin
gampin
```

rule 5 metathesis

In [ ]:

```
meta = hfst.regex(" [t p] -> [p t], [l b]-> [b l],[ n m ] -> [m n]|| ?* V _
apply_rules('atipin',[nasal_ng , nasal_m, o2u, Syncope, silence, meta])
```

```
atipin
atipin
atipin
atipin
atpin
atpin
aptin
```

In [ ]:

```
apply_rules('talaban',[nasal_ng , nasal_m, o2u, Syncope, silence, meta])
```

```
talaban
talaban
talaban
talaban
talban
talban
tablan
```

the missing is tablin

In [ ]:

```
apply_rules('talabin',[nasal_ng , nasal_m, o2u, Syncope, silence, meta])
```

```
talabin
talabin
talabin
talabin
talbin
talbin
tablin
```

In [ ]:

```
apply_rules('tanimin',[nasal_ng , nasal_m, o2u, Syncope, silence, meta])
```

```
tanimin
tanimin
tanimin
tanimin
tanmin
tanmin
tamnin
```

```
In [ ]: apply_rules('lamanin',[nasal_ng , nasal_m, o2u, Syncope, silence, meta])
```

```
lamanin
lamanin
lamanin
lamanin
lamnin
lamnin
lamnin
```

the rules are perfectly set now.

```
In [ ]: defs['nang'] = nasal_ng
        defs['nam'] = nasal_m
        defs['meta'] = meta
        defs['silence'] = silence
        defs['ou'] = o2u
        defs['Syncope'] = Syncope
        defs.keys()
```

```
Out[ ]: dict_keys(['C', 'V', 'S', 'nasal_ng', 'nasal_m', 'meta', 'silence', 'o2u', '
        Syncope', 'nang', 'nam', 'ou'])
```

```
In [ ]: TagalogPHON = hfst.regex('nang .o. nam .o.  ou .o. Syncope .o. silence .o. m
```

```
In [ ]: apply_rules('lamanin',[TagalogPHON])
```

```
lamanin
lamnin
```

```
In [ ]: defs['TagalogPHRASE'] = TagalogPHRASE
        defs['TagalogMOR'] = TagalogMOR
        defs['TagalogPHON'] = TagalogPHON
```

```
In [ ]: Tagalog = hfst.regex('TagalogPHRASE .o. TagalogMOR .o. TagalogPHON', definit
```