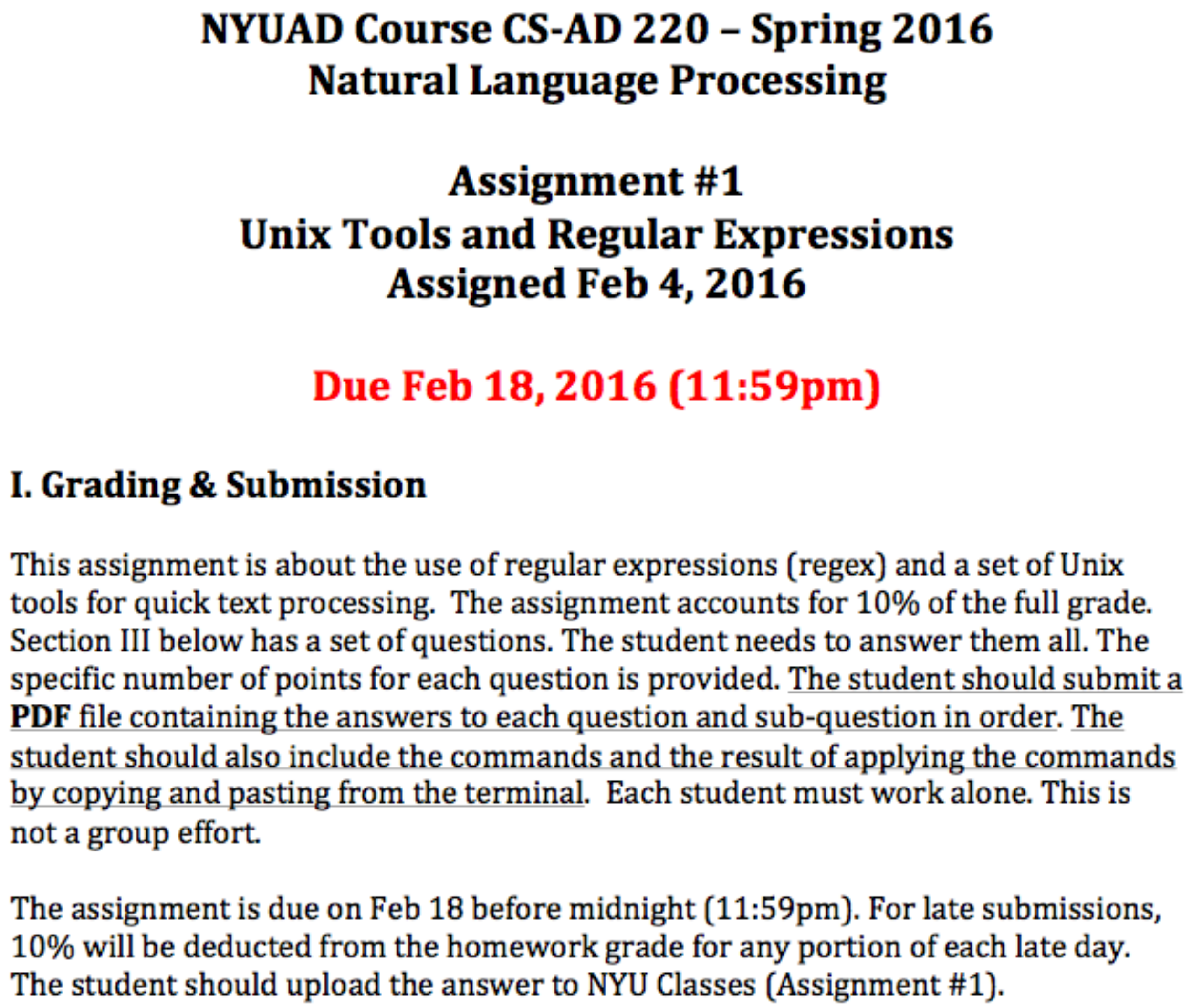
CS-AD 220 – Spring 2016

Natural Language Processing

Session 7: 18-Feb-16

Prof. Nizar Habash

Some slides are adapted from Jurafsky and Martin's course slides on Speech and Language Processing



NYUAD Course CS-AD 220 – Spring 2016
Natural Language Processing

Assignment #1
Unix Tools and Regular Expressions
Assigned Feb 4, 2016

Due Feb 18, 2016 (11:59pm)

I. Grading & Submission

This assignment is about the use of regular expressions (regex) and a set of Unix tools for quick text processing. The assignment accounts for 10% of the full grade. Section III below has a set of questions. The student needs to answer them all. The specific number of points for each question is provided. The student should submit a **PDF** file containing the answers to each question and sub-question in order. The student should also include the commands and the result of applying the commands by copying and pasting from the terminal. Each student must work alone. This is not a group effort.

The assignment is due on Feb 18 before midnight (11:59pm). For late submissions, 10% will be deducted from the homework grade for any portion of each late day. The student should upload the answer to NYU Classes (Assignment #1).

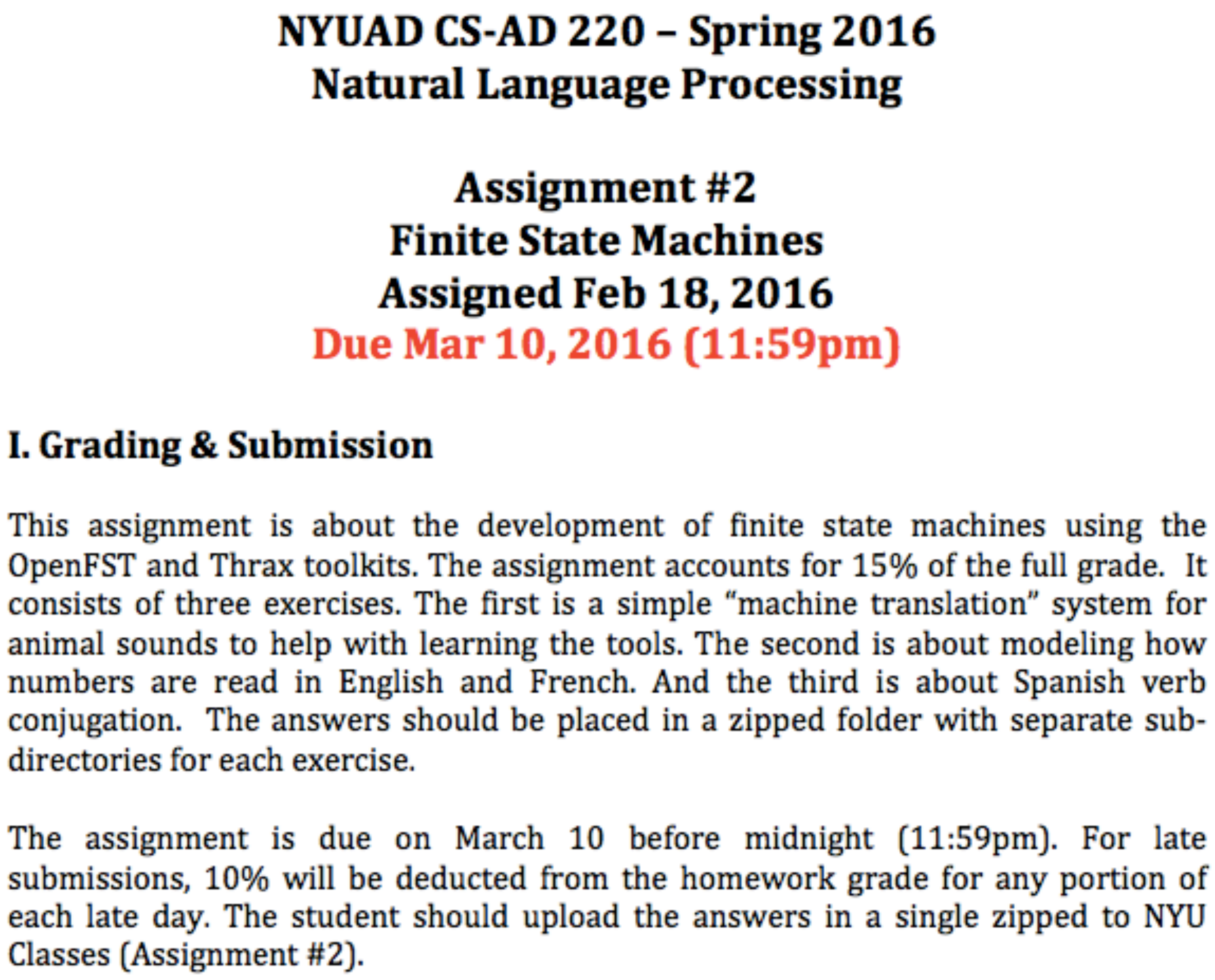
Assignment #1 posted on NYU Classes

Moving Legislative Day Class

- Spring Break is March 18 – 25, 2016
- Sat March 26, 2016 is a Legislative *Thursday*
- Move to

Sat April 2, 2016 at 10am

Same Classroom C2-E049



NYUAD CS-AD 220 – Spring 2016
Natural Language Processing

Assignment #2
Finite State Machines
Assigned Feb 18, 2016
Due Mar 10, 2016 (11:59pm)

I. Grading & Submission

This assignment is about the development of finite state machines using the OpenFST and Thrax toolkits. The assignment accounts for 15% of the full grade. It consists of three exercises. The first is a simple "machine translation" system for animal sounds to help with learning the tools. The second is about modeling how numbers are read in English and French. And the third is about Spanish verb conjugation. The answers should be placed in a zipped folder with separate sub-directories for each exercise.

The assignment is due on March 10 before midnight (11:59pm). For late submissions, 10% will be deducted from the homework grade for any portion of each late day. The student should upload the answers in a single zipped to NYU Classes (Assignment #2).

Assignment #2 posted on NYU Classes

Computational Morphology

- Components of a morphological processor
  - Lexicon
    - The list of morphemes (stems, affixes, etc.) together with basic information
      - main categories (noun, verb, adjective, …)
      - sub-categories (regular noun, irregular noun, …)
  - Morphotactics
    - The model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word.
  - Orthographic/Phonological Rules
    - The model of changes that occur in a word (normally when morphemes combine).

Morpholgy and FSAs

- We’d like to use the machinery provided by FSAs to capture these facts about morphology
  - Accept strings that are in the language
  - Reject strings that are not
  - And do so in a way that doesn’t require us to in effect list all the words in the language

Simple FSM for English Plurals

- We want to model
  - Regular plurals: cat+PL → cats
  - Regular plurals with rewrite: fox+PL → foxes
  - Irregular plurals: goose+PL → geese
- Multi-tape Finite State Machines

	cat	**fox**	**goose**
Lexical	cat+PL	fox+PL	goose+PL
Intermediate	cat^s	fox^s	geese
Surface	cats	foxes	geese

- We want the singulars (+SG) to be handled also

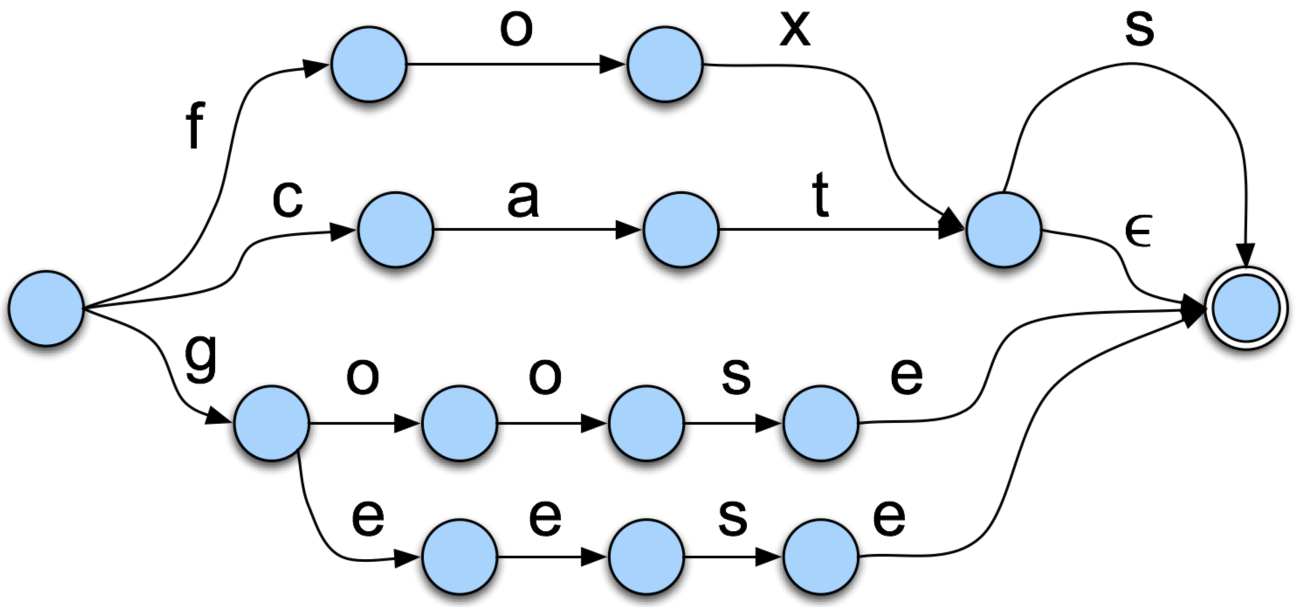
Start Simple

- Regular singular nouns are ok
- Regular plural nouns have an -s on the end
- Irregulars are ok as is

Simple Rules

$$\begin{tikzcd}
q_0 \arrow[r, bend left, "\text{reg-noun}"] \arrow[rr, bend right=20, "\text{irreg-pl-noun}"'] \arrow[rr, bend right=50, "\text{irreg-sg-noun}"'] & q_1 \arrow[r, bend left, "\text{plural -s}"] & q_2
\end{tikzcd}$$

Now Plug in the Words



Parsing/Generation vs. Recognition

- We can now run strings through these machines to recognize strings in the language
- But recognition is usually not quite what we need
  - Often if we find some string in the language we might like to assign a structure to it (parsing)
  - Or we might have some structure and we want to produce a surface form for it (production/generation)
- Example
  - From "cats" to "cat +N +PL"

Applications

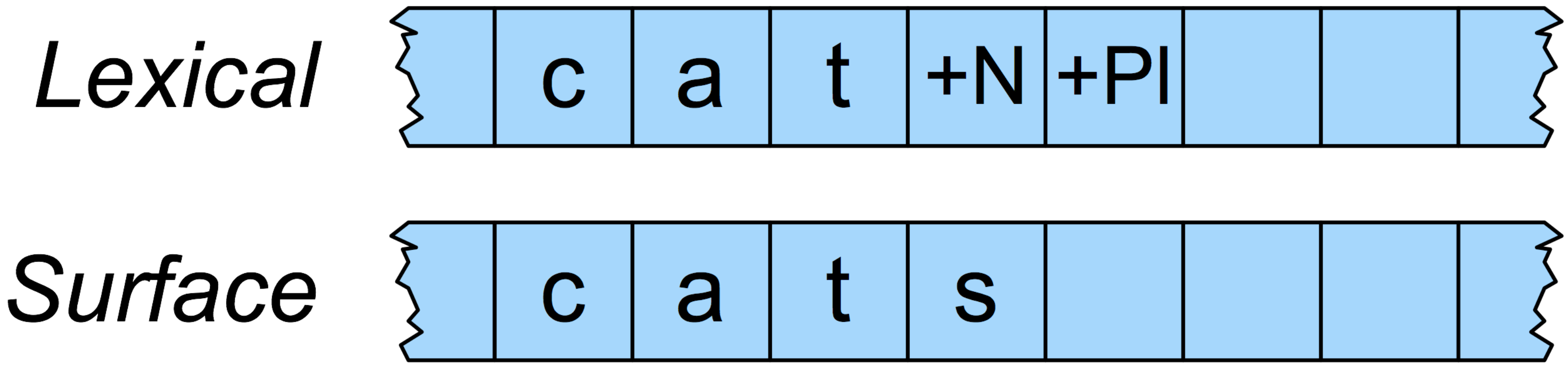
- The kind of parsing we' re talking about is normally called morphological analysis
- It can either be
  - An important stand-alone component of many applications (spelling correction, information retrieval)
  - Or simply a link in a chain of further linguistic analysis

Finite State Transducers

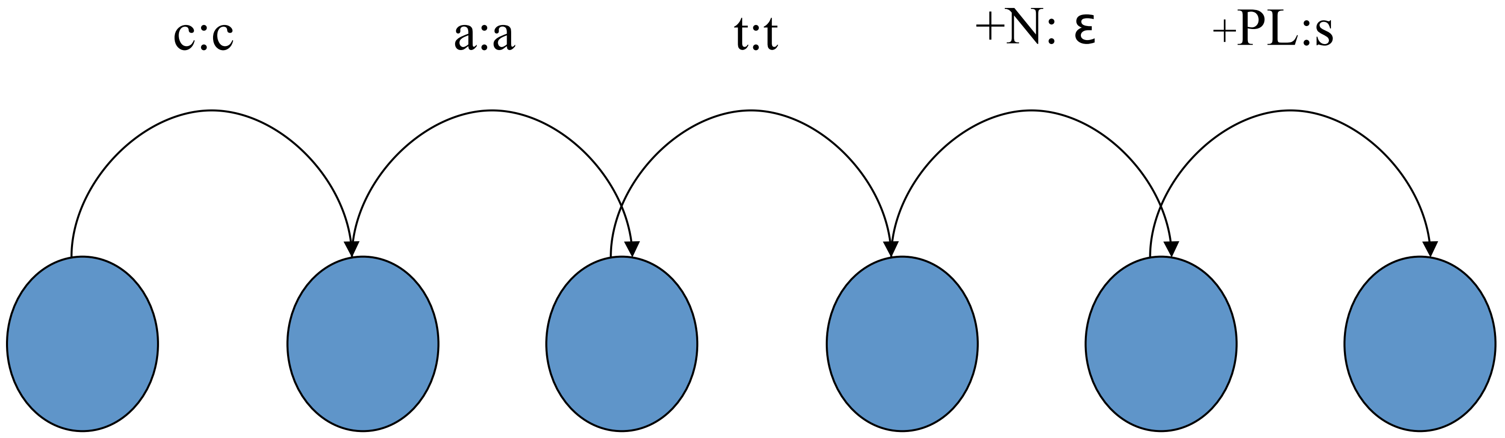
- The simple story
  - Add another tape
  - Add extra symbols to the transitions

  - On one tape we read "cats", on the other we write "cat +N +PL"

Finite State Transducers



Finite State Transducers



- c:c means read a c on one tape and write a c on the other
- +N:ε means read a +N symbol on one tape and write nothing on the other
- +PL:s means read +PL and write an s

Ambiguity

- Recall that in non-deterministic recognition multiple paths through a machine may lead to an accept state.
  - Didn't matter which path was actually traversed
- In FSTs the path to an accept state does matter since different paths represent different parses and different outputs will result

Ambiguity

- What’s the right parse (segmentation) for
  - Unionizable
  - Union-ize-able
  - Un-ion-ize-able
- Each represents a valid path through the derivational morphology machine.

Ambiguity

- There are a number of ways to deal with this problem
  - Simply take the first output found
  - Find all the possible outputs (all paths) and return them all (without choosing)
  - Bias the search so that only one or a few likely paths are explored

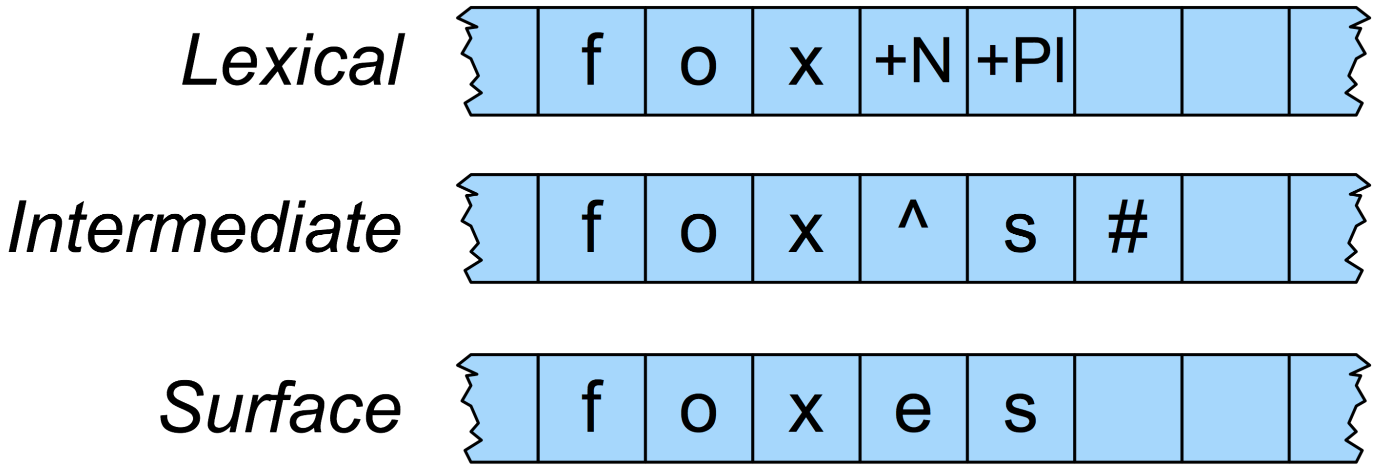
The Gory Details

- Of course, its not as easy as
  - "cat +N +PL" <-> "cats"
- As we saw earlier there are geese, mice and oxen
- But there are also a whole host of spelling/ pronunciation changes that go along with inflectional changes
  - Cats vs Dogs
  - Fox and Foxes

Multi-Tape Machines

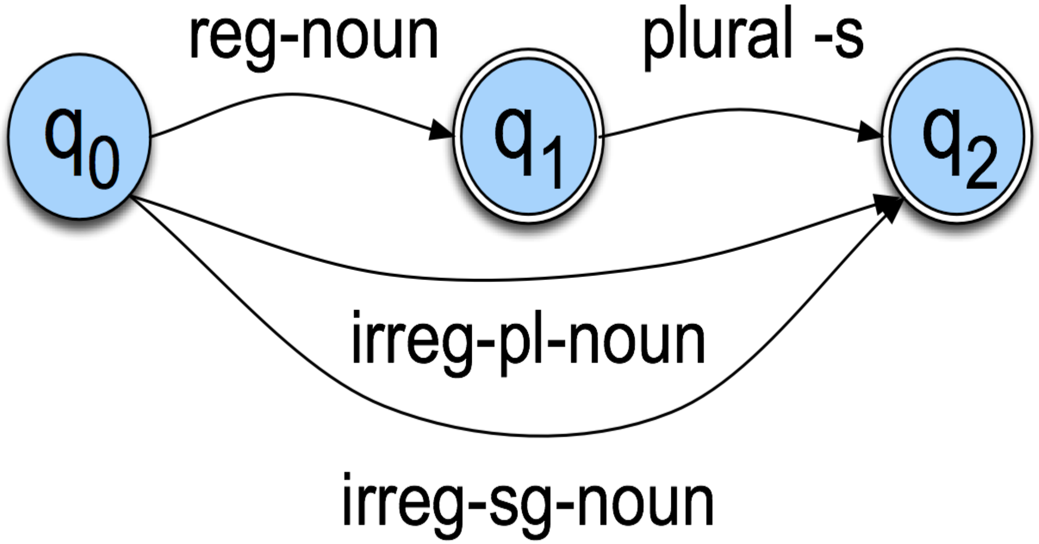
- To deal with these complications, we will add more tapes and use the output of one tape machine as the input to the next
- So to handle irregular spelling changes we'll add intermediate tapes with intermediate symbols

Multi-Level Tape Machines



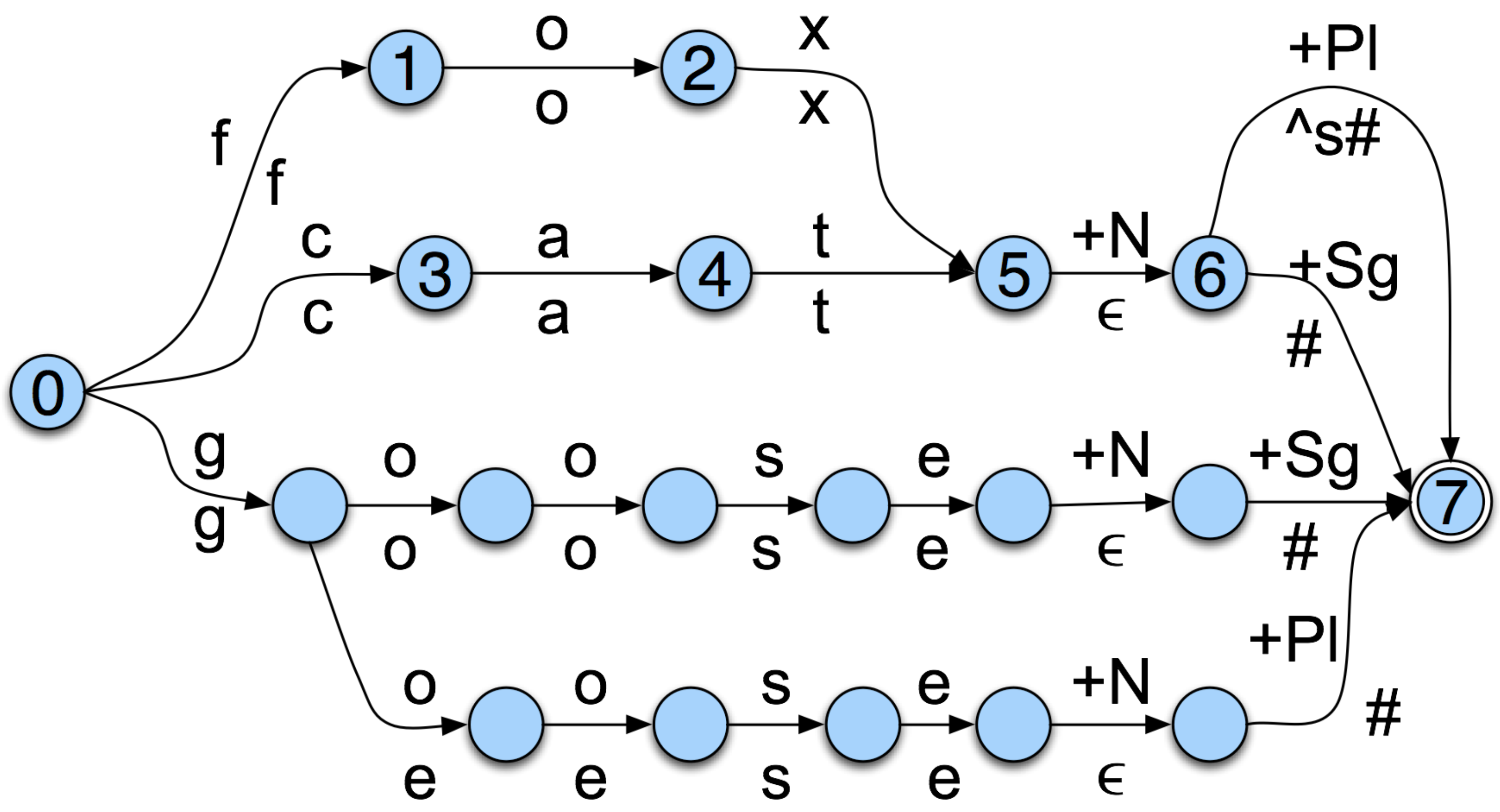
- We use one machine to transduce between the lexical and the intermediate level, and another to handle the spelling changes to the surface tape

Lexical to Intermediate Level



- Lexicon specifies
  - Regular Nouns = {cat, fox}
  - Irregular Singular Nouns = {goose}
  - Irregular Plural Nouns = {geese}

Lexical to Intermediate Level



Intermediate to Surface

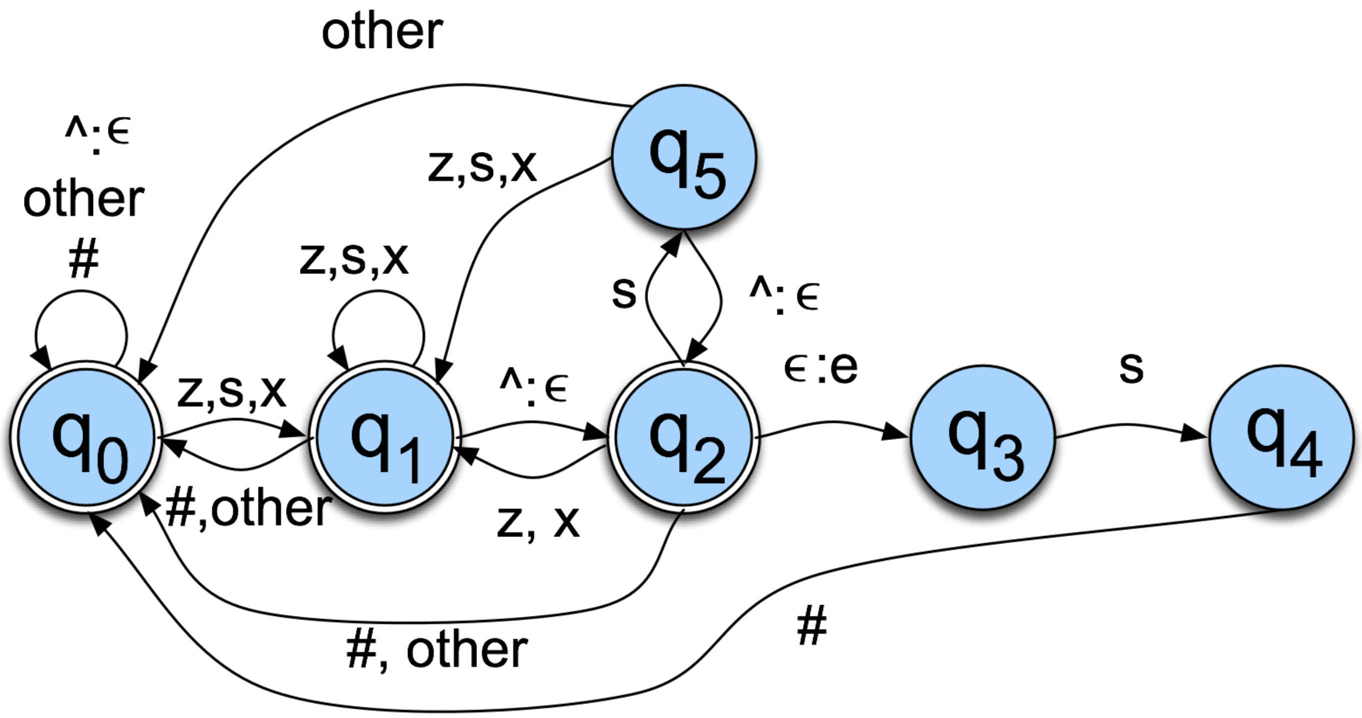
- The add an "e" rule as in fox^s# <-> foxes#

E-insertion rule

$\varepsilon \rightarrow e$ / {x,s,z} ^ __ s#

^-deletion rule

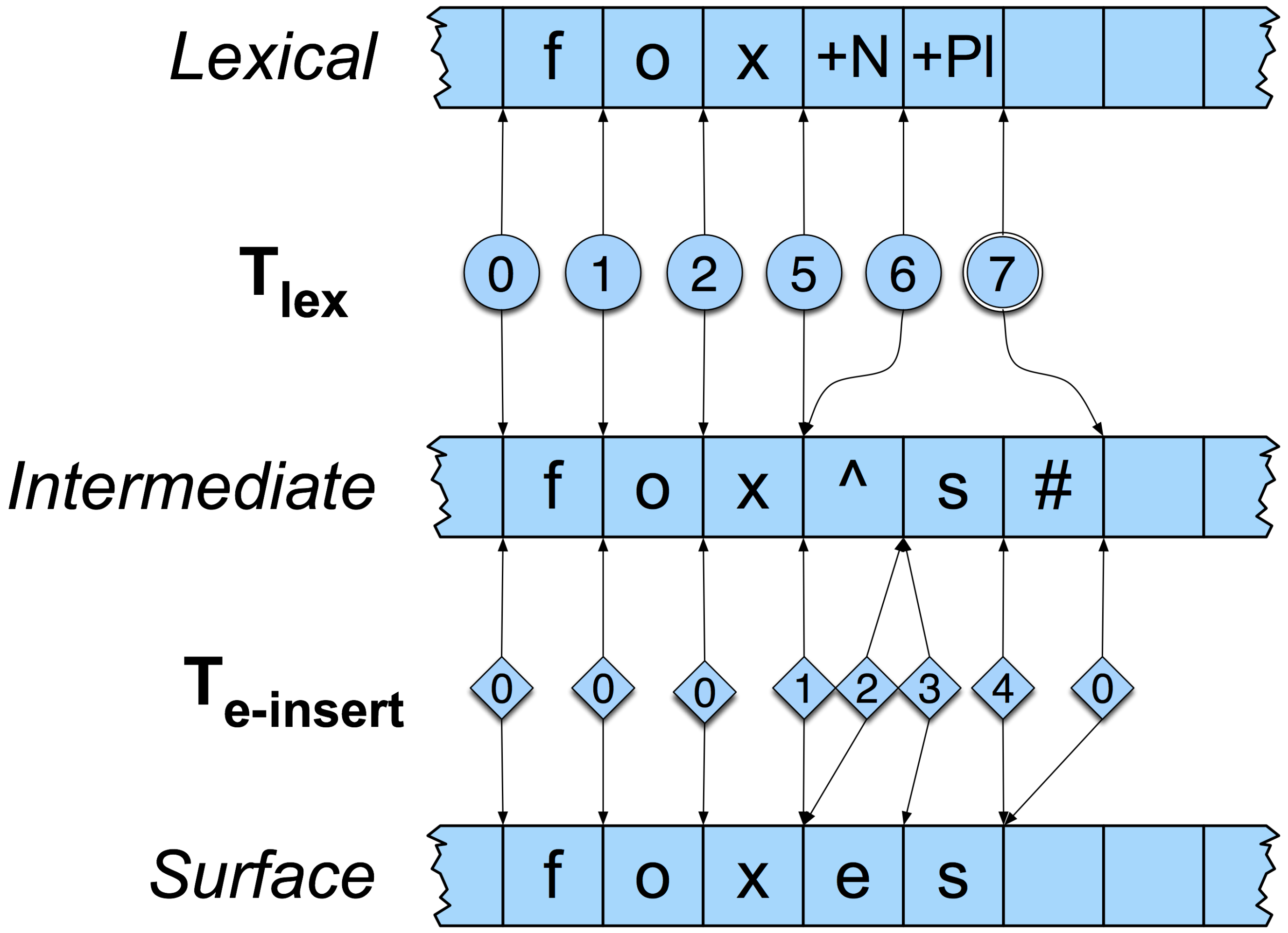
$\text{^} \rightarrow \varepsilon$



Note

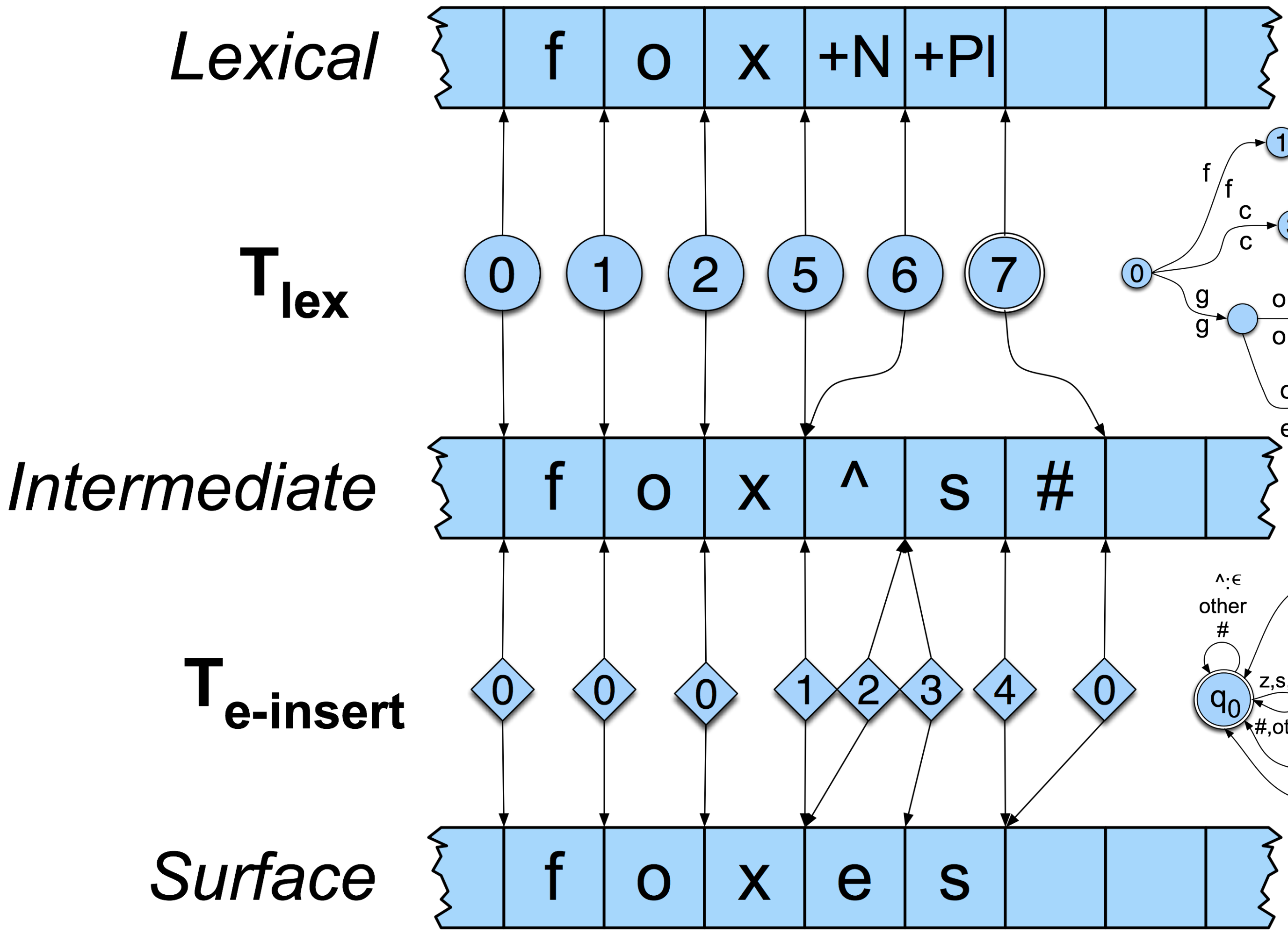
- A key feature of this machine is that it doesn’t do anything to inputs to which it doesn’t apply.
- Meaning that they are written out unchanged to the output tape.

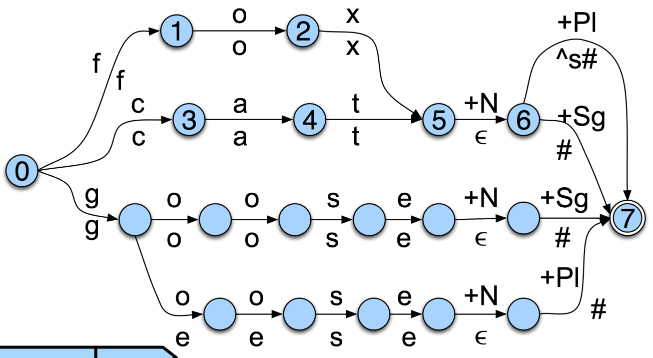
Foxes

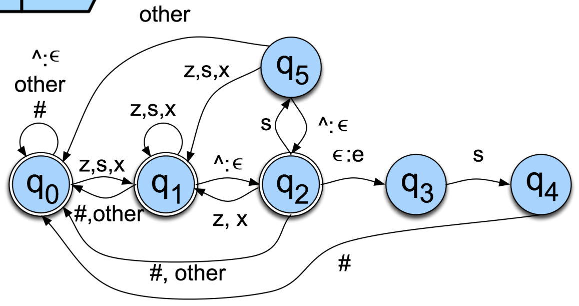


Lexical
f
o
x
+N
+Pl
T lex
0
1
2
5
6
7
Intermediate
f
o
x
^
s
#
T e-insert
0
0
0
1
2
3
4
0
Surface
f
o
x
e
s

Foxes


Lexical
f o x +N +Pl
T_lex
0 1 2 5 6 7
Intermediate
f o x ^ s #
T_e-insert
0 0 0 1 2 3 4 0
Surface
f o x e s

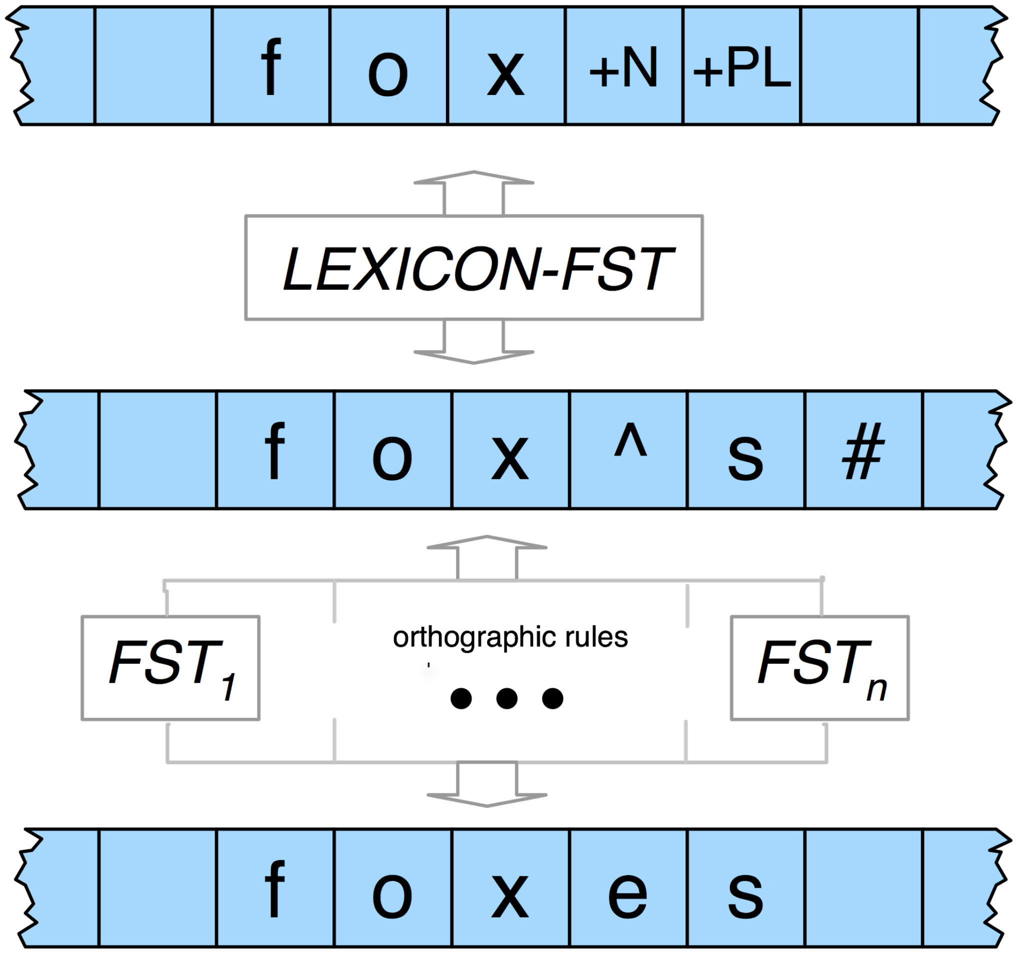

0 1 2 3 4 5 6 7
f f
o o
x x
c c
a a
t t
+N ϵ
+Pl ^s#
+Sg #
g g
o o
o o
s s
e e
+N ϵ
+Sg #
o e
o e
s s
e e
+N ϵ
+Pl #


q0 q1 q2 q3 q4 q5
other
^:ϵ
other
#
z,s,x
z,s,x
z,s,x
#,other
^:ϵ
z, x
s
^:ϵ
ϵ:e
s
#, other
#

Overall Scheme

- We now have one FST that has explicit information about the lexicon (actual words, their spelling, facts about word classes and regularity).
  - Lexical level to intermediate forms
- We have a larger set of machines that capture orthographic/spelling rules.
  - Intermediate forms to surface forms

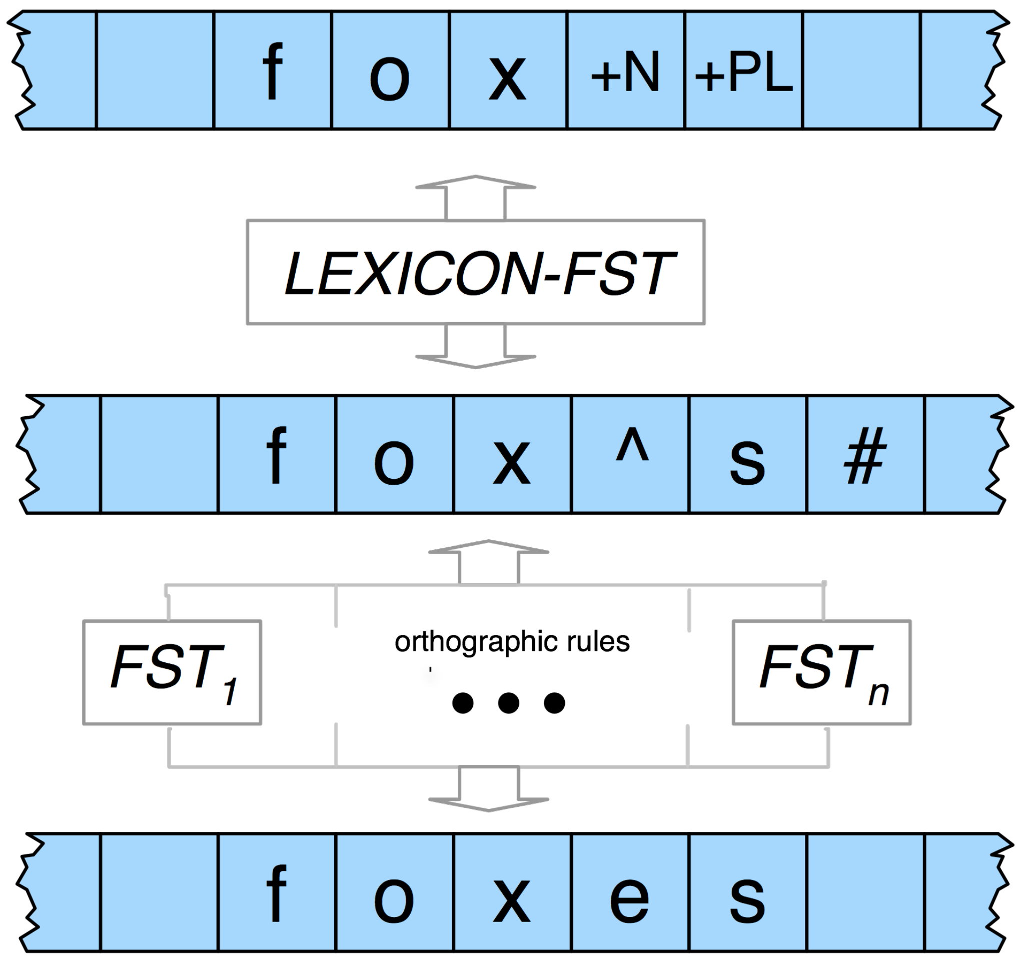
Overall Scheme



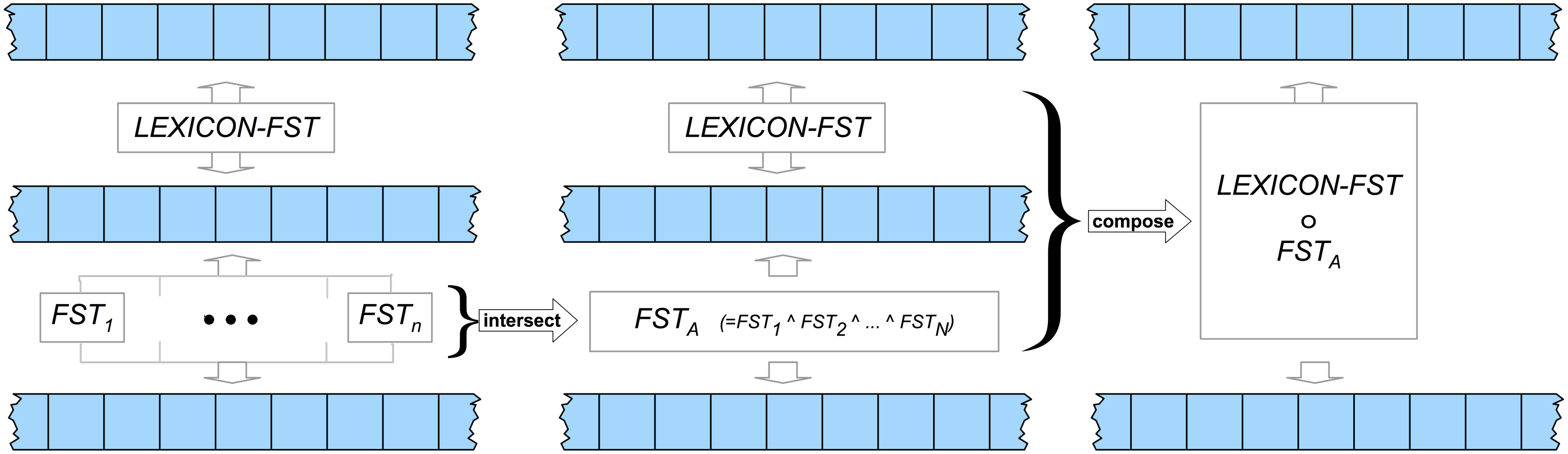
Cascades

- This is an architecture that we' ll see again and again
    - Overall processing is divided up into distinct rewrite steps
    - The output of one layer serves as the input to the next
    - The intermediate tapes may or may not wind up being useful in their own right

Overall Plan



Final Scheme

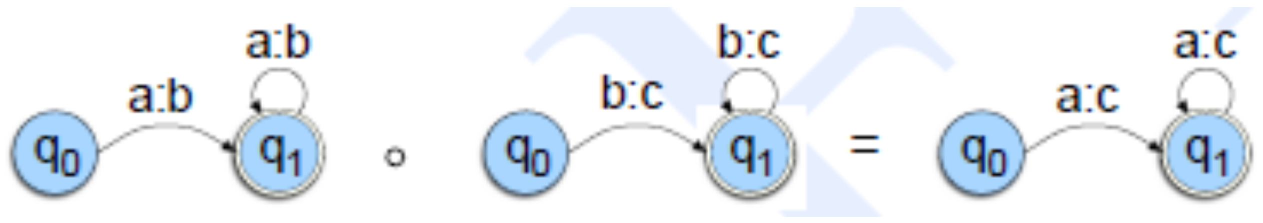


Composition

1. Create a set of new states that correspond to each pair of states from the original machines (New states are called (x,y), where x is a state from M1, and y is a state from M2)
2. Create a new FST transition table for the new machine according to the following intuition…

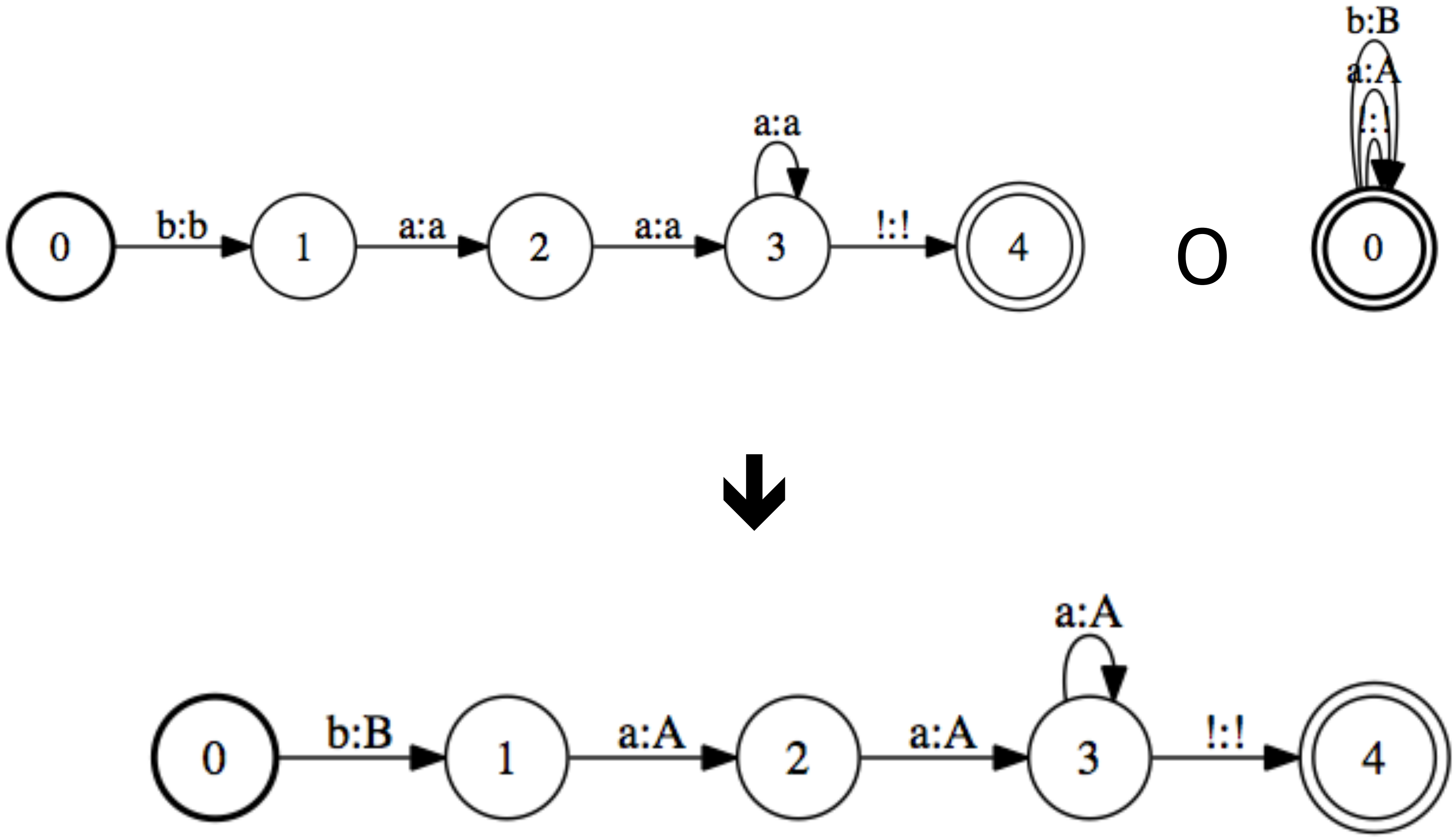
Composition

- There should be a transition between two states in the new machine if it’s the case that the output for a transition from a state from M1, is the same as the input to a transition from M2 or…



- The composition operator is represented with o
- X = A o B
- X is the result of A composing with B

Composition

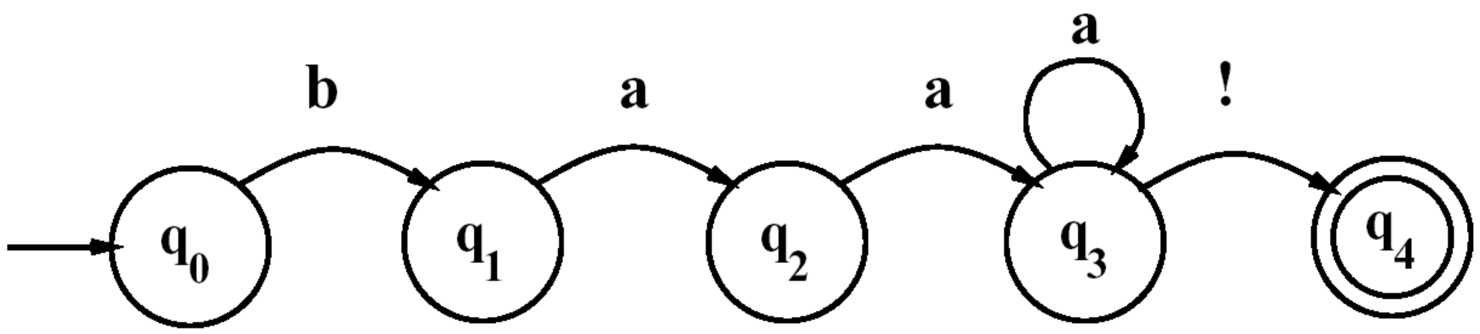


OpenFST

- **OpenFST** is a library for constructing, combining, optimizing, and searching weighted finite-state transducers
  - http://www.openfst.org/twiki/bin/view/FST/WebHome

- The OpenGrm **Thrax** Grammar Compiler is a set of tools for compiling grammars expressed as regular expressions and context-dependent rewrite rules into weighted FSTs
  - http://openfst.cs.nyu.edu/twiki/bin/view/GRM/Thrax

Building a simple FST



- FSA is equivalent to an FST where input and output are the same.
- <start> <end> <input> <output>

```
0     1     b     b
1     2     a     a
2     3     a     a
3     3     a     a
3     4     !     !
4                        <<<<accepting state!
```

Exercise

- Build a sheep language capitalization machine. It should only accept sheep words.
  - baaaaa! → BAAAAA!
  - Baba! → <REJECT>

- We'll do it in OpenFST and in Thrax

OpenFST

- Symbol file = {a,A,b,B,!}
- Sheep grammar = baa+!
- Capitalization: a→A, b→B, !→!
- Input test = baaaaaaaa!

Files

```
cat sheep.sym
<epsilon>       0
a       1
A       2
b       3
B       4
!       5
cat sheep.txt
0       1       b       b
1       2       a       a
2       3       a       a
3       3       a       a
3       4       !       !
4
cat cap.txt
0       0       b       B
0       0       a       A
0       0       !       !
0
```

```
cat sheep.word.txt
0   1   b   b
1   2   a   a
2   3   a   a
3   4   a   a
4   5   a   a
5   6   a   a
6   7   a   a
7   8   a   a
8   9   a   a
9   10  !   !
10
```

Compile

```
fstcompile --isymbols=sheep.sym --osymbols=sheep.sym
--keep_isymbols --keep_osymbols sheep.txt sheep.fsa

fstcompile --isymbols=sheep.sym --osymbols=sheep.sym
--keep_isymbols --keep_osymbols cap.txt cap.fst

fstcompile --isymbols=sheep.sym --osymbols=sheep.sym
--keep_isymbols --keep_osymbols sheep.word.txt
sheep.word.fsa
```

Compose

```
fstcompose sheep.fsa cap.fst sheep.cap.fst

fstprint sheep.cap.fst
0     1     b     B
1     2     a     A
2     3     a     A
3     3     a     A
3     4     !     !
4
```

Compose 2

```
fstcompose sheep.word.fsa sheep.cap.fst |fstprint
```

```
0     1     b     B
1     2     a     A
2     3     a     A
3     4     a     A
4     5     a     A
5     6     a     A
6     7     a     A
7     8     a     A
8     9     a     A
9     10    !     !
10
```

Same result with:

```
fstcompose sheep.word.fsa sheep.fsa | fstcompose –
cap.fst  |fstprint
```

Thrax Version

```
cat sheep.grm
   sheep = "b" "a" "a"+ "!”;
   cap = ( ("a":"A") | ("b":"B") | "!" )*;
   export sheepcap = Optimize[ Compose[ sheep, cap] ] ;

thraxmakedep --save_symbols sheep.grm && make
```

This creates an FST archive (.far) file

```
perl ../testFAR.pl sheep.far sheepcap
baaa!
      baaa!  BAAA!
baaaaaa!
      baaaaaa!     BAAAAAA!
```

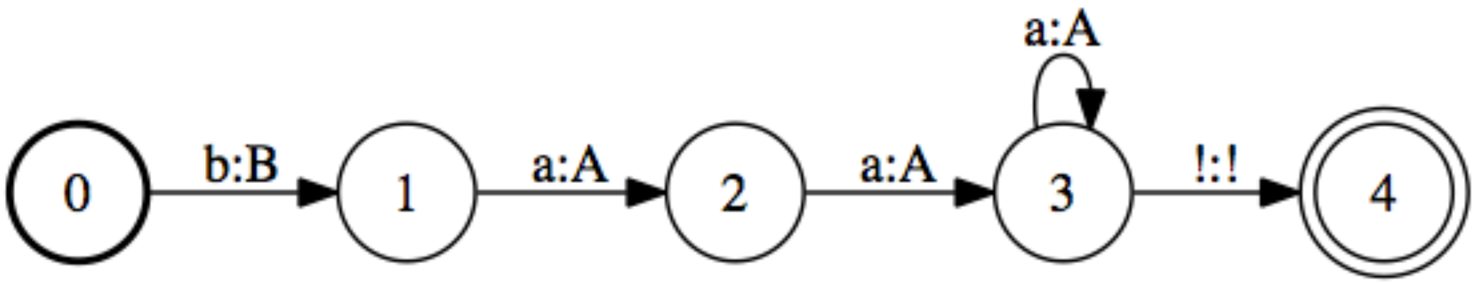
FSTs in FARs

```
farextract --filename_suffix=.fst sheep.far

fstprint sheepcap.fst
0     1     b     B
1     2     a     A
2     3     a     A
3     4     !     !
3     3     a     A
4
```

Visualizing FSTs

```
fstdraw sheepcap.fst | dot –Tpdf > sheepcap.pdf
or
fstdraw sheepcap.fst | dot –Tps > sheepcap.ps
```



English Plurals Thrax File (1/2)

```
###########################################################
# English Plurals
###########################################################

###########################################################
#Vocabulary
###########################################################

letter = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m"
       | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z";

alpha = letter | "P" | "L" | "S" | "G" | "+" | "-" | "^" ;

alpha_star = alpha*;

###########################################################
#Lexicon
###########################################################

reg_noun = ( "cat" | "fox" );
irreg_sg_noun = ("goose" );
irreg_pl_noun = ("goose" : "geese");
```

English Plurals Thrax File (2/2)

```
##########################################################
#Lexical to intermediate
##########################################################
lex2inter = reg_noun ("+SG":"")
           | reg_noun ("+PL":"^s")
           | irreg_sg_noun ("+SG" : "")
           | irreg_pl_noun ("+PL" : "")
;

##########################################################
#Rewrite rules
##########################################################
e_insert = CDRewrite["" : "e", ("x"|"s"|"z") "^","s",alpha_star];
delete_morph_boundary = CDRewrite["^" : "","","",alpha_star];

word = lex2inter @ e_insert @ delete_morph_boundary;

##########################################################
#Final
##########################################################
export generate = Optimize[ word ] ;
export analyze  = Invert [generate] ;

##########################################################
```

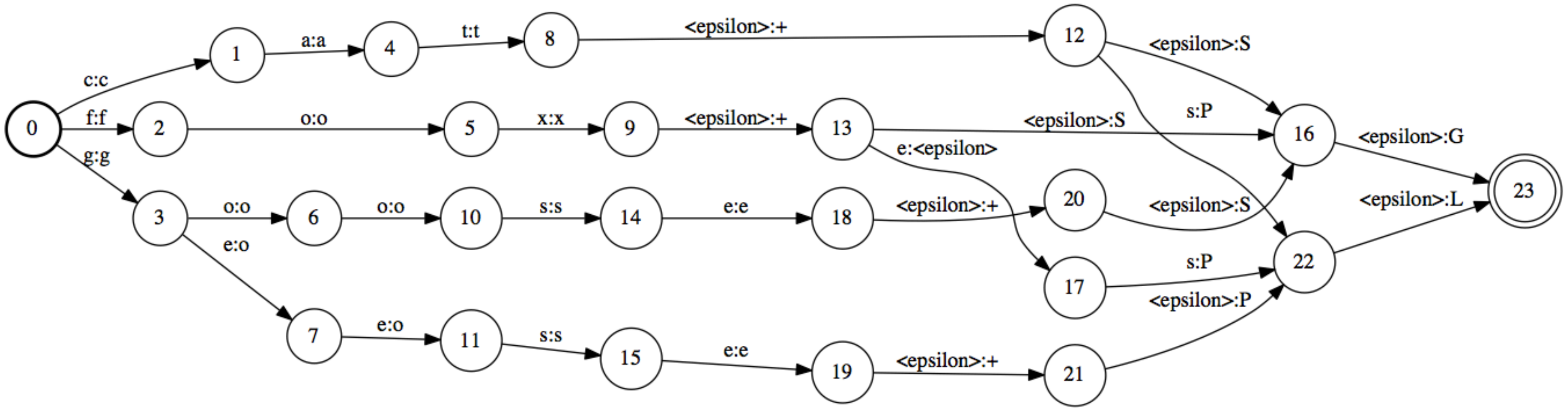
From Thrax Manual

- **CDRewrite**
  - given a transducer and two context acceptors (and the alphabet machine), this will generate a new FST that performs the rewrite everywhere in the provided contexts.
- CDRewrite[tau, lambda, rho, sigma_star]
  - Tau = transducer
  - Lambda = left context
  - Rho = right context
  - Sigma_star = Alphabet machine (<alphabet>*)

  Returns an FST

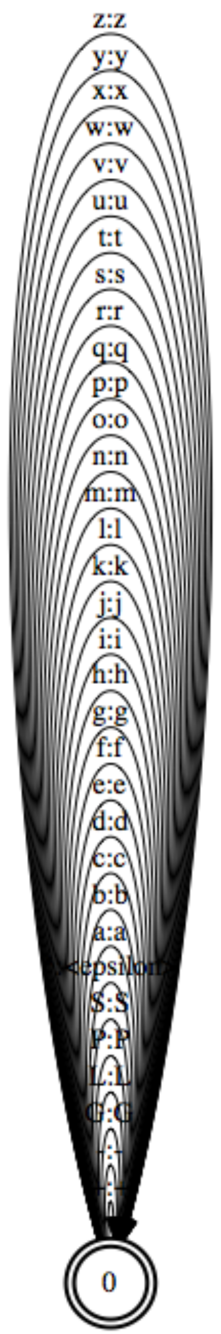
-
  ```
  e_insert = CDRewrite[ "" : "e",
                        ("x"|"s"|"z") "^",
                        "s",
                        alpha_star];
  ```

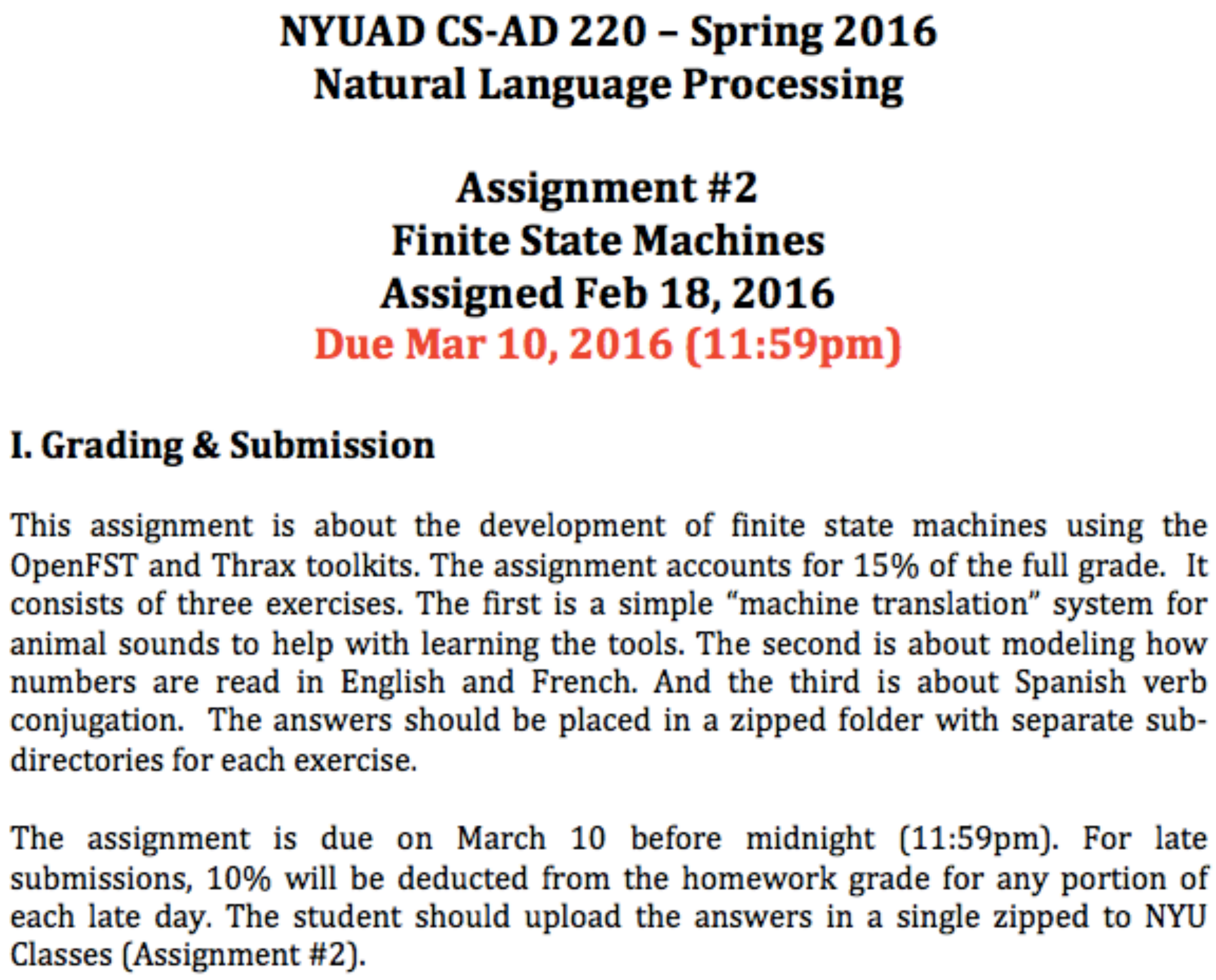
analyze FST


0
c:c
f:f
g:g
1
2
3
a:a
4
t:t
8
<epsilon>:+
12
o:o
5
x:x
9
<epsilon>:+
13
o:o
6
o:o
10
s:s
14
e:e
18
<epsilon>:+
20
e:o
7
e:o
11
s:s
15
e:e
19
<epsilon>:+
21
<epsilon>:S
s:P
<epsilon>:S
e:<epsilon>
16
<epsilon>:S
17
s:P
<epsilon>:P
22
<epsilon>:G
<epsilon>:L
23

delmorbound FST

```
fstprint delmorbound.fst
0	0	+	+
0	0	-	-
0	0	G	G
0	0	L	L
0	0	P	P
0	0	S	S
0	0	^	<epsilon>
0	0	a	a
0	0	b	b
0	0	c	c
0	0	d	d
0	0	e	e
0	0	f	f
0	0	g	g
0	0	h	h
0	0	i	i
0	0	j	j
0	0	k	k
0	0	l	l
0	0	m	m
0	0	n	n
0	0	o	o
0	0	p	p
0	0	q	q
0	0	r	r
0	0	s	s
0	0	t	t
0	0	u	u
```





NYUAD CS-AD 220 – Spring 2016
Natural Language Processing

Assignment #2
Finite State Machines
Assigned Feb 18, 2016
Due Mar 10, 2016 (11:59pm)

I. Grading & Submission

This assignment is about the development of finite state machines using the OpenFST and Thrax toolkits. The assignment accounts for 15% of the full grade. It consists of three exercises. The first is a simple "machine translation" system for animal sounds to help with learning the tools. The second is about modeling how numbers are read in English and French. And the third is about Spanish verb conjugation. The answers should be placed in a zipped folder with separate sub-directories for each exercise.

The assignment is due on March 10 before midnight (11:59pm). For late submissions, 10% will be deducted from the homework grade for any portion of each late day. The student should upload the answers in a single zipped to NYU Classes (Assignment #2).

Assignment #2 posted on NYU Classes

Next Time

- Read J+M Chap 3 (3.8 to end); NH Chap 4 (intro and 4.1 only)

- Assignment #1 due Feb 18 (TODAY) before midnight!