Words (Still)

Parts of Speech

What is a POS?

- Words vary in their syntactic distribution
- Words vary in their morphology
- Words vary in their semantics

A part of speech captures the similarities between words as a linguistic representation

POS

Noun/Pronoun:

- Syntactic → follow a determiner
- Morphology → rule for pluralization
- Semantics → convey person, place, thing, or idea

For example,

- A cat slept on him.

POS

Verb:

- Semantic → convey events
- Syntactic → preceeded by a noun
- Morphology → rule for changing tense

For example,

- Joan defenestrated her ex’s clothes

POS

Determiner/Quantifier

Semantic → Denotes a quantity of referent

Syntactic →Usually precedes a noun

Morphology → Nothing special :P

For example,

The monkey ate some bananas.

POS

Adjective:

Syntactic → precedes a noun

Morphology → positive, comparative, and superlative forms

Semantics → specifies properties of the noun

For example,

The devastated contestant left quickly.

POS

Adverb:

Syntactic → precedes a verb

Morphology → many have the -ly ending

Semantics → describes properties of the verb

For example,

A antelope quickly ran past the photographer.

POS

Preposition

- Semantics → Denotes a relation
- Syntactic →Precedes a noun phrase
- Morphology → Nothing special :p

POS Activity

Running

A

Hedgehog

Beyond

Technicolor

Swiftly

It

POS Activity

Running	Verb
A	Determiner
Hedgehog	Noun
Beyond	Preposition
Technicolor	Adjective
Swiftly	Adverb
It	Pronoun

Stemmers

What are stemmers?

Rules that cut off strings.

For example,

Chop ‘ness’: ‘Thoughtfullness’ → ‘thoughfull’

Chop ‘full’: ‘Thoughtfull’ → ‘thought’

Problems with stemmers

For example,

Chop ‘full’: ‘full’ → ‘ ’

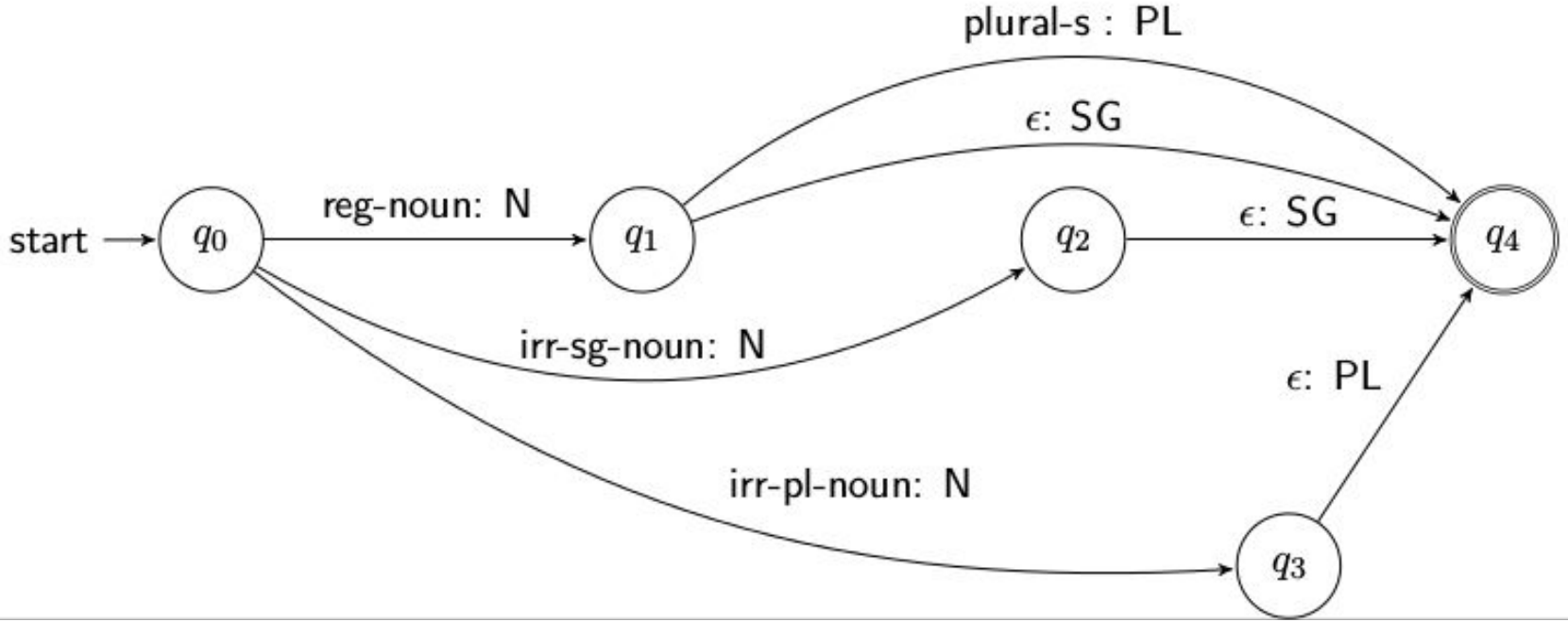
Chop ‘ing’: ‘sing’ → ‘s’

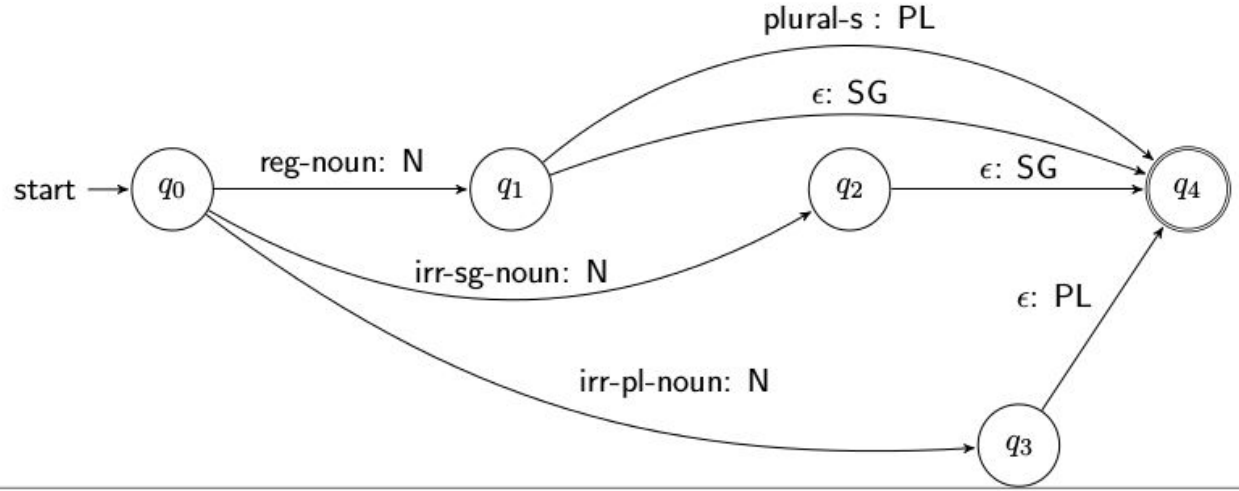
Lemmatizers

What are lemmatizers?

- The FSAs we looked at yesterday are lemmatizers.
- Rather than manipulate strings, they verify and parse morphemes.

FST Lemmatizer






```
1 q0([X|L1],[n|L2]):- reg_noun(X), q1(L1,L2).
2 q1([X|L1], [pl|L2]):- plural_s(X), q4(L1,L2).
3 q1(L1, [sg|L2]):-  q4(L1,L2).
4 q0([X|L1],[n|L2]):- irr_sg_noun(X), q2(L1,L2).
5 q2(L1,[sg|L2]):- q4(L1,L2).
6 q0([X|L1],[n|L2]):- irr_pl_noun(X), q3(L1,L2).
7 q3(L1,[pl|L2]):- q4(L1,L2).
8 q4([],[]).
```

FST Lemmatizer (Chaves, 2014)

Greedy Parsing

The Algorithm

1) X = ϵ (the empty string)
2) Read one character from string
3) X’ = X + Y
4) If there is a W such that

   W is longest word in dictionary that matches X’

   X = X’

   Go to 2,

   Else

   return X

```
greed([],[]).
greed(String,[Token|TokenList]):-
      tokenize([],String,StringRest,Token),
      greed(StringRest,TokenList).

tokenize(String,[Char|Rest1],Rest2,Token):-
   append(String,[Char],NewString),
   check(NewString,Rest1,Rest2,Token).

check(NewString,Rest,Rest,NewString):- token(NewString).
check(NewString,Rest1,Rest2,Token):-
  tokenize(NewString,Rest1,Rest2,Token).

token([t,h,e]).
token([t,h,i,s]).
token([p,i,n]).
token([p,i,n,e]).
```

Greedy Parsing Algorithm in Prolog (Chaves, 2014)

```
?- greed([t,h,e,p,i,n],TokenList).
TokenList = [[t, h, e], [p, i, n]]

?- greed([t,h,e,p,i,n,e],TokenList).
TokenList = [[t, h, e], [p, i, n, e]]

?- greed([t,h,i,s,p,i,n,e,p,i,n],TokenList).
TokenList = [[t, h, i, s], [p, i, n, e], [p, i, n]]

?- greed([p,i,n,t,h,i,s,p,i,n,e],TokenList).
TokenList = [[p, i, n], [t, h, i, s], [p, i, n, e]]

?- greed([t,h,e,i,n,e],TokenList).
false.
```

(Chaves, 2014)