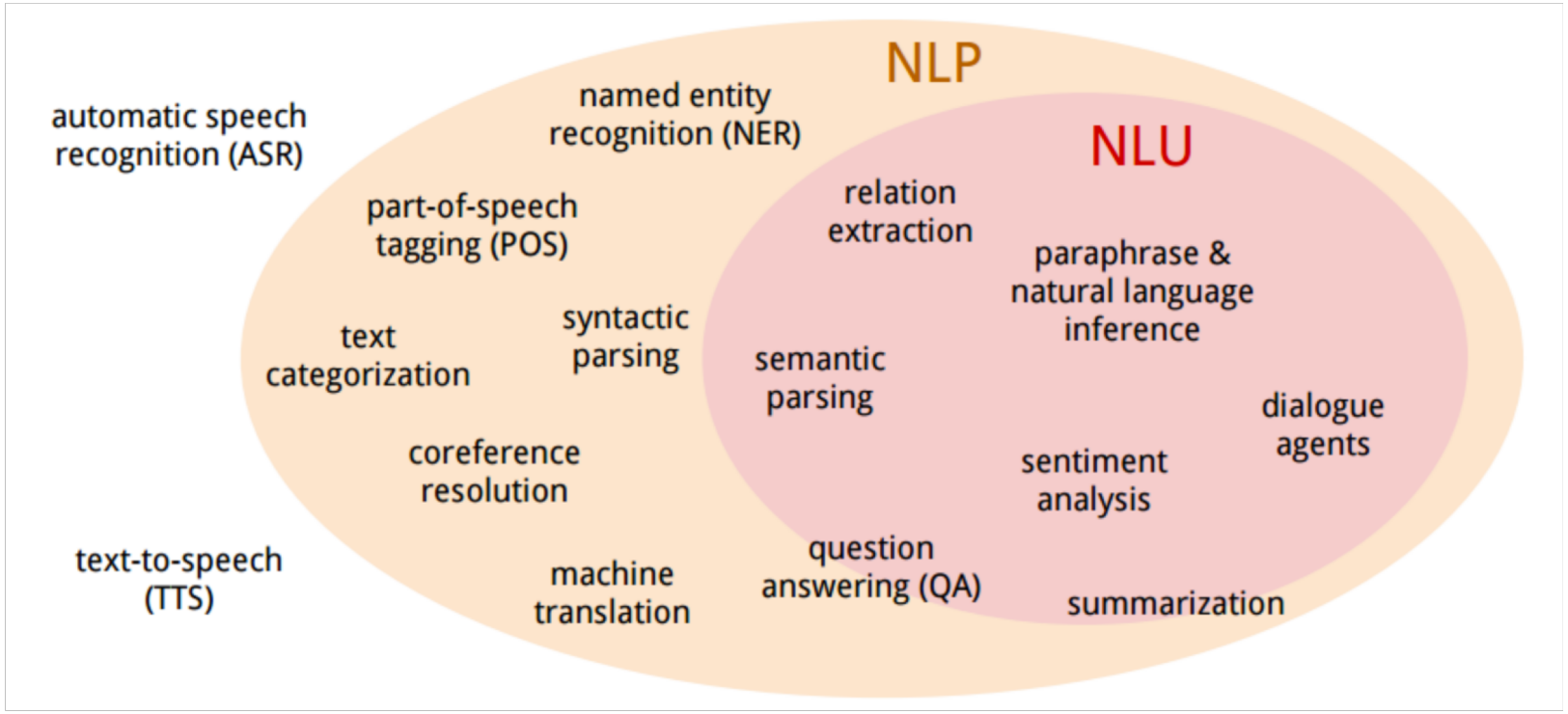
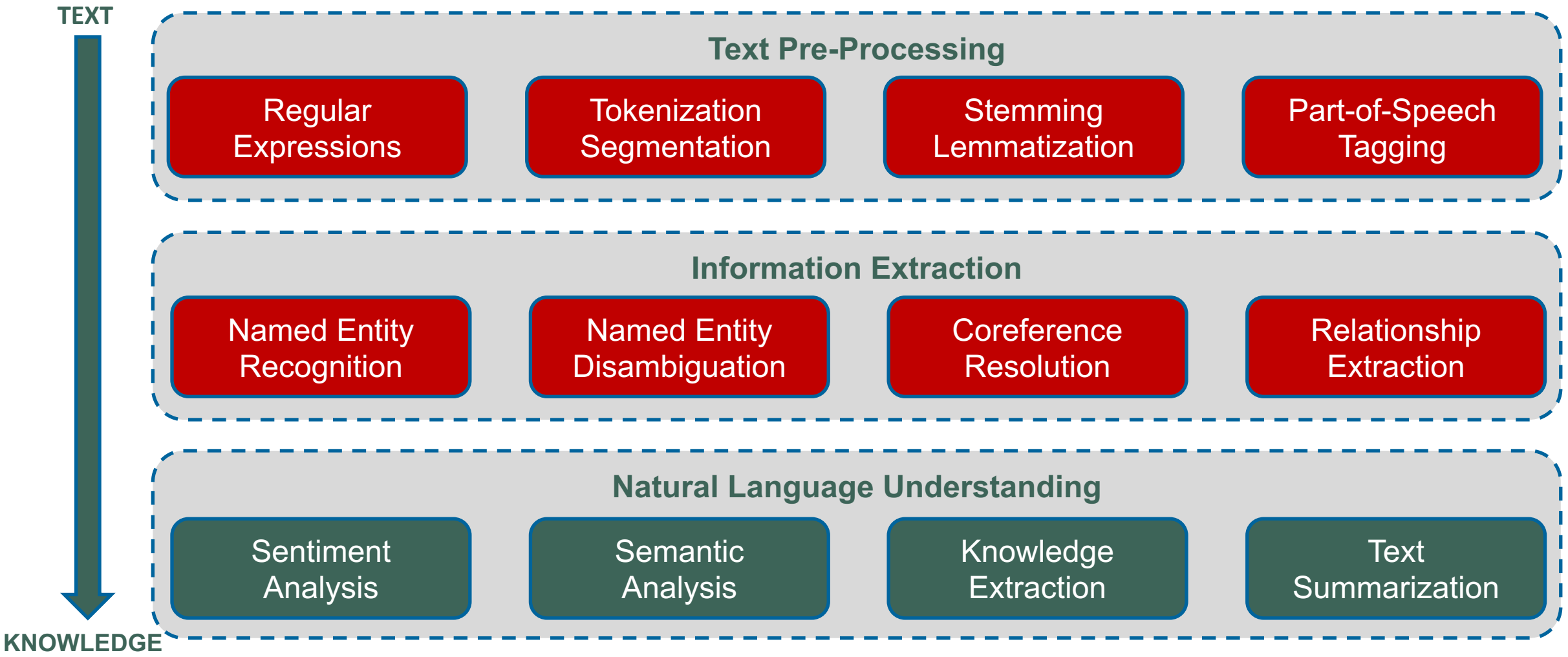
BASIC CONCEPTS

Natural Language Processing (NLP) is a field of computer science and artificial intelligence, concerned with making computers process natural (human) language



Computational Linguistics (CL) is the field of using computers to understand language

OVERVIEW OF THE BASIC NLP PIPELINE



TEXT PRE-PROCESSING: REGULAR EXPRESSIONS

A formal language for specifying text strings

- Used as the first model for any text processing
- Used as features in machine learning classifiers

Find all instances of the word "*the*" in a text

`the` – misses capitalized examples

`[tT]he` – Incorrectly returns *other* or *theology*

`[^a-zA-Z][tT]he[^a-zA-Z]`

False Positives: matching `there,then,other`
False Negatives: not matching `The`

Pattern	Matches	Examples
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`
`[^A-Z]`	Not an upper case letter	`Oyfn pripetchik`
`[^Ss]`	Neither 'S' nor 's'	`I have no exquisite reason"`
`[^e^]`	Neither e nor ^	`Look here`
`a^b`	The pattern a carat b	`Look up a^b now`

SENTENCE SEGMENTATION

!, ? are relatively unambiguous

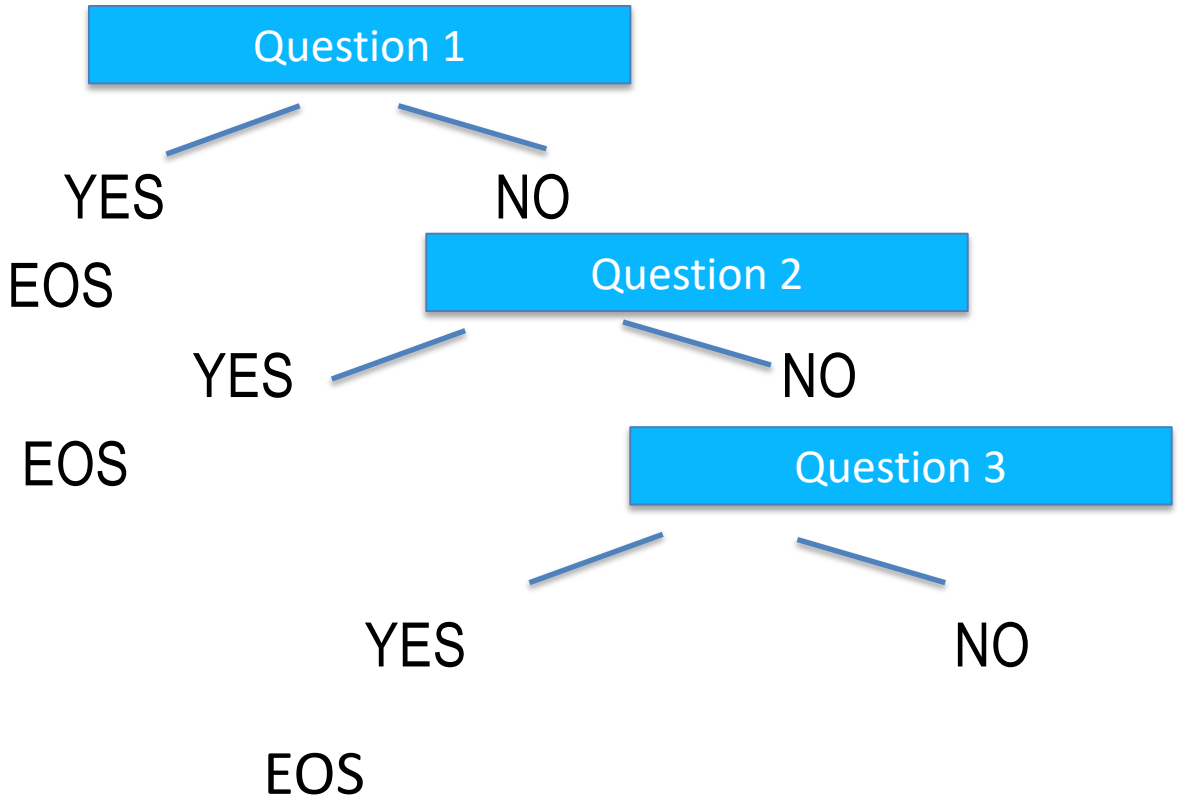
Period "." is quite ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Build a binary classifier

- Looks at a "."
- Decides EndOfSentence/NotEndOfSentence
- Classifiers: hand-written rules, regular expressions, or machine-learning

Decision Trees – Build a Rule for End of Sentence



SENTENCE SEGMENTATION

!, ? are relatively unambiguous

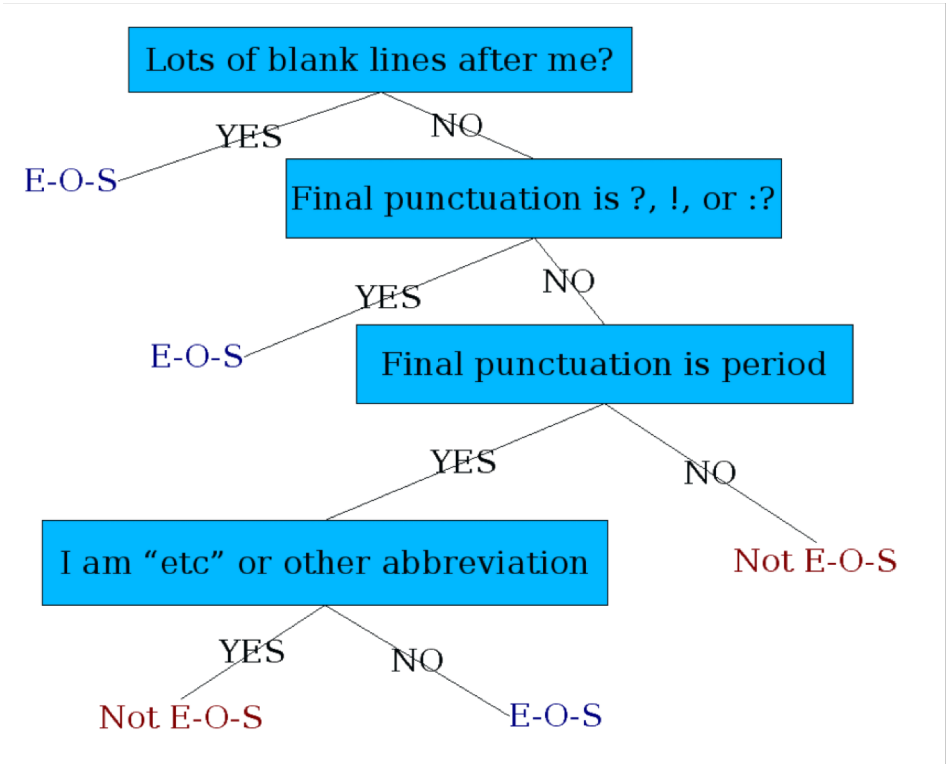
Period "." is quite ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Build a binary classifier

- Looks at a "."
- Decides EndOfSentence/NotEndOfSentence
- Classifiers: hand-written rules, regular expressions, or machine-learning

Decision Trees



TOKENIZATION

Tokenization is the task of chopping up a character sequence up into *tokens*, i.e. words, *n*-grams, sentences

***n*-gram** is a contiguous sequence of *n* items from a given sequence of text

Stop words are the most common words that are of little value in search query

To be, or not to be:	**To be , or not to be**
Unigrams:	**to be or not to be**
Bigrams:	**to be or not to be**
Trigrams:	**to be or not to be**
Stopwords:	**to be or not**

```
>>> from nltk.tokenize import sent_tokenize, word_tokenize
>>> sent_tokenize(s)
['Good muffins cost $3.88\nin New York.', 'Please buy me\ntwo of them.', 'Thanks.']
>>> [word_tokenize(t) for t in sent_tokenize(s)]
[['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.'],
['Please', 'buy', 'me', 'two', 'of', 'them', '.'], ['Thanks', '.']]
```

STEMMING & LEMMATIZATION

- **Stemming** – reducing inflected or derived words to their base form
- **Lemmatization** – grouping together different inflected forms of a word
- Both are linguistic morphology terms with the same goal but different approaches

```
>>> wordnet_lemmatizer.lemmatize('are')
'are'
>>> wordnet_lemmatizer.lemmatize('is')
'is'
```

```
>>> wordnet_lemmatizer.lemmatize('is', pos='v')
u'be'
>>> wordnet_lemmatizer.lemmatize('are', pos='v')
u'be'
```

```
>>> from nltk.stem.porter import PorterStemmer
>>> porter_stemmer = PorterStemmer()
>>> porter_stemmer.stem('maximum')
u'maximum'
>>> porter_stemmer.stem('presumably')
u'presum'
>>> porter_stemmer.stem('multiply')
u'multipli'
```

```
>>> from nltk.stem import WordNetLemmatizer
>>> wordnet_lemmatizer = WordNetLemmatizer()
>>> wordnet_lemmatizer.lemmatize('dogs')
u'dog'
>>> wordnet_lemmatizer.lemmatize('churches')
u'church'
>>> wordnet_lemmatizer.lemmatize('aardwolves')
u'aardwolf'
```

PART-OF-SPEECH (POS) TAGGING

POS Tagging, aka grammatical tagging or word-category disambiguation, is the process of marking a word in a text with a particular part of speech, based on both its definition and context

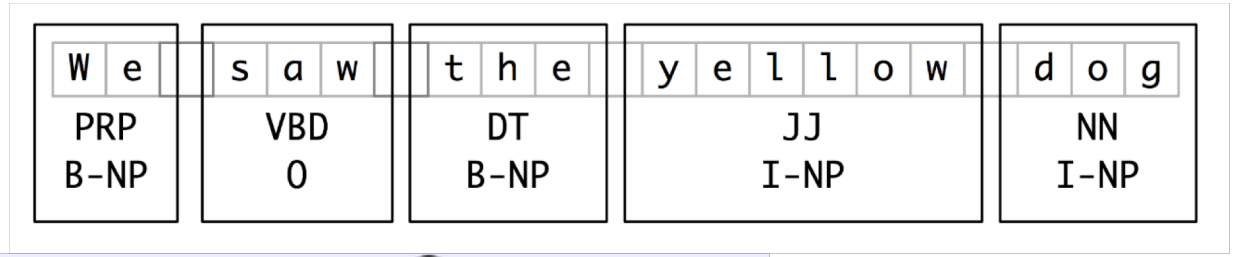
```
>>> import nltk
>>> text = nltk.word_tokenize("Dive into NLTK: Part-of-speech tagging and POS Tagger")
>>> text
['Dive', 'into', 'NLTK', ':', 'Part-of-speech', 'tagging', 'and', 'POS', 'Tagger']
>>> nltk.pos_tag(text)
[('Dive', 'JJ'), ('into', 'IN'), ('NLTK', 'NNP'), (':', ':'), ('Part-of-speech', 'JJ'), ('tagging', 'NN'),
('and', 'CC'), ('POS', 'NNP'), ('Tagger', 'NNP')]
```

PennTreeBank

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

CHUNKING – SHALLOW PARSING

Segment and label multi-token sequences using regular expressions and POS tags




```
>>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),  ❶
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("cat", "NN")]

>>> grammar = "NP: {<DT>?<JJ>*<NN>}"  ❷

>>> cp = nltk.RegexpParser(grammar)  ❸
>>> result = cp.parse(sentence)  ❹
>>> print(result)  ❺
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
>>> result.draw()  ❻
```