

Lecture 2

July 26, 2016

Linux, Cluster Computing, RegEx



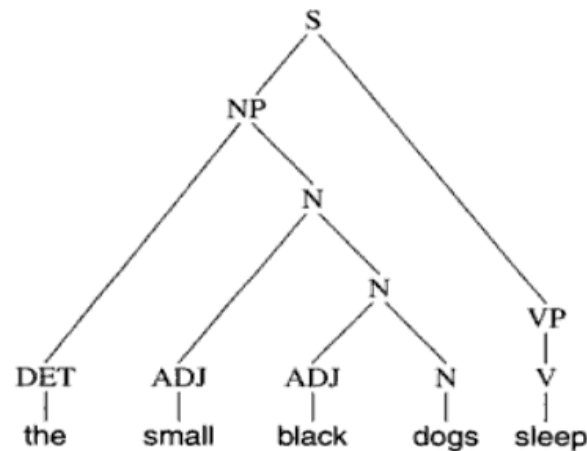
Reminder: start the recording

Assignment 1

- Due before class on This Thursday– 4:30 p.m.
- If you submit work early and want early grading, write a note in *CollectIt* that your submission is final.

Constituents

- Can sentences be analyzed as containing sub-units which consist of one or more words?



(Hausser 1998)

This is a phrase structure tree.

Hypothesis: The class of grammatical constituents is closed.

Constituent Types

- Constituents are characterized by the part-of-speech of their main word, the “head.”
- Thus, we notice that for many languages, a sentence comprises a:
 - subject (a Noun Phrase or NP)
 - predicate (a Verb Phrase or VP)
- These may be composed of other constituents
 - Prepositional phrase (PP)
 - Determiner phrase (DP)

Constituent Construction

- Noun phrases (NPs)

(DET NN)	<i>The ostrich</i>
(NNP)	<i>Kim</i>
(NN NN)	<i>container ship</i>
(DET JJ NN)	<i>A purple lawnmower</i>
(DET JJ NN)	<i>That darn cat</i>

- Verb phrases (VPs)

(VB) tango	
(VBD NP NP)	<i>gave the dog a bone</i>
(VBD NP PP)	<i>gave a bone to the dog</i>

Syntax

The set of rules governing permissible constructions in a language.

- Syntax constrains the ways in which words may be combined to form constituents and sentences.
- Syntax forms one part of the description, or *grammar*, of a language.



Prescriptive v. descriptive grammar

- Prescriptive
 - Rules against certain usages. Few if any rules for what *is* allowed.
 - Prepositions are not for ending sentences with.
- Descriptive
 - Rules characterizing what people *do* say.
 - Goal is to characterize all and only what speakers find acceptable.
 - Based on the scientific method

Slide: Emily Bender

Artificiality of prescriptive rules

- Fill in the blanks: *he/his, they/their, or something else?*

Everyone insisted that___record was unblemished.

Everyone drives___own car to work.

Everyone was happy because___passed the test.

Everyone left the room, didn't___?

Everyone left early. ___ seemed happy to get home.

Slide: Bender, Sag, Wasow 2003

Two kinds of ambiguity

1. Lexical ambiguity

The bank is crumbling.



?



Two kinds of ambiguity

2. Structural ambiguity

I saw a man with a telescope.



?



?



Is that all?



I (often) saw a man with a telescope.

Ambiguity

Q: What kind of ambiguity does the following sentence illustrate?

Have that report on my desk by Friday.

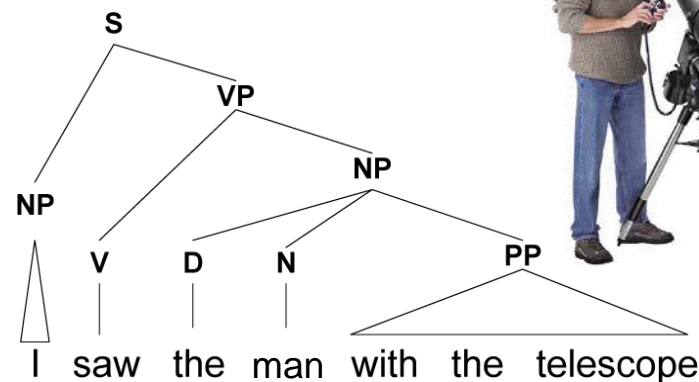
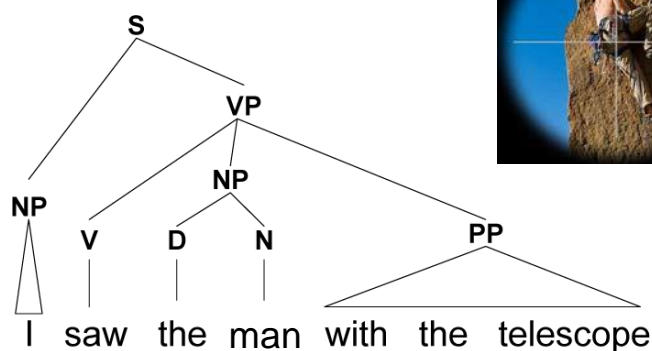
A: Both structural and lexical ambiguity.



In English, prosody in speech provides disambiguation by, for example, marking topic and focus.

Constituents

- Constituents help us understand and classify syntactic structure
- Here, phrase structure trees show us how constituent structure helps us characterize ambiguity



Trees: Dan Jinguji

Analytical NLP: Summary

- Language has significant complexity
- Ambiguity is an inherent feature of language
- Inferring the syntactic rules of a language is difficult
- It appears that syntactic rules *are* amenable to computational approximation, but there are a great number of them, and they are subtle
- See next slide for the obligatory optimism...

Latest Work: Hybrid analytical/statistical systems

- Examples:

- Stephan Oepen, Erik Velldal, Jan Tore Lønning, Paul Meurer, Victoria Rosén, and Dan Flickinger. 2007. “Towards hybrid quality-oriented machine translation”

<http://www.mt-archive.info/TMI-2007-Oepen.pdf>

- Parse ranking in the English Resource Grammar
- Unsupervised learning of rules
 - Poon and Domingos 2009 “Unsupervised Semantic Parsing” (UW CSE)

<http://www.aclweb.org/anthology-new/D/D09/D09-1001.pdf>

Statistical Natural Language Processing

- Large gains in practical application of stochastic methods in the past 15 years
- Example: MT
 - Microsoft Bing Translator
 - Google translate
 - GIZA++/Moses SMT toolkit
- Insight: clever math—with limited linguistic motivation—can work surprisingly well



These improvements are not just due to Moore's law

The mid-1990s

- Advances in computing power
- Perceived lack of progress in analytical NLP
- What if the “rules” of linguistics as we intuitively imagine them are too hard (or too incorrect, or too biased...) to capture?
- Let’s forget any preconceived notion of what the rules *should* be and use math to characterize what the rules *appear* to be in practice.
- German *Verbmobil* project had success with statistical machine translation (This large project also developed rule-based systems)

Source: Koehn 2010 “Statistical Machine Translation”

Corpus Linguistics

The study of language as expressed in samples (corpora) or “real world” text.

- Wikipedia

This can be thought of as a variant of analytical NLP where the form, substance, and quantity of “rules” are generated automatically according to the maximization of an empirical objective function.

Treebank

- A collection of text with syntactic structure annotation for each sentence
 - Human-annotated
 - Machine generated by precision grammar
- Penn Treebank (PTB) (Marcus et al. 1994)
<http://www.cis.upenn.edu/~treebank/>
- Linguistic Data Consortium (LDC)
<http://www ldc.upenn.edu/>



PTB Example

```
( (S (NP (NNP John) )  
    (VP (VBZ loves)  
        (NP (NNP Mary) ) )  
    (. .) ) ) )
```

PTB tag set

CC	Coordinating conjunction	PRP	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential there	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	to
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VCN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non 3rd person singular present
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP	Possessive wh-pronoun
PRP	Personal pronoun	WRB	Wh-adverb

Counting

- Corpus linguistics is essentially about counting and arranging collections and sequences of elements (words, characters).
- Elements are collected together into sets or bags
 - This can be done in a number of ways as we'll see
- It's important to keep track of what we are counting. We can count:
 - The number of elements in a collection
 - This is fairly straightforward. This is called the “cardinality” of the collection
 - The number of occurrences of a particular element in a collection
 - We can distinguish this by calling it a “tally.”
 - The number of collections of a certain type that are found in a corpus
 - The number of collections of a certain type that could possibly be formed from another collection of elements

Collections

- Consider a collection of n elements. It can be ordered or unordered, and distinct (all elements are unique), or with repeated elements.

	Ordered	Unordered
Distinct	(i.e. MRU cache)	vocabulary, alphabet, set
Not-Distinct	sentence, string, vector, sequence, word	bag, multiset



We distinguish ordering from sorting. Ordering refers to whether the order matters in the modeling context, not whether the elements happen to be in some sorted order for practical (programming) purposes

Distinctness

- Distinct, i.e. no repetition (“set,” “vocabulary”)

```
{ apple, pear, banana, orange, pomelo }
```

we can count:

- The number of distinct elements (“cardinality,” “vocabulary size”)

- With repetition (“bag” or “multiset”)

```
{ apple, apple, banana, apple }
```

we can count:

- The number of distinct elements
- The number of times each element appears (“tally” or “multiplicity”)

- In mathematics, the word ‘set’ generally means unordered and distinct unless otherwise specified
- Parentheses are usually used when order matters (n-tuples), braces are used when order doesn’t matter { sets }.

Ordering

Ordered, ("string," "vector," "sequence," or "n-tuple")

```
( a, man, gave, sandy, a, pomelo )  
                               ≠  
( a, a, gave, kim, pomelo, sandy )
```

Unordered, Not-distinct ("bag")

```
{ a, man, gave, sandy, a, pomelo }  
                               =  
{ sandy, pomelo, man, gave, a, a }
```

Unordered, Distinct ("vocabulary")

```
{ a, man, gave, sandy, pomelo }
```

- Distinct collections where the order matters are less common. An example is the set of (distinct) words used in a document, by order of appearance. This is a form of Most Recently Used (MRU) cache.



Vocabulary

The distinct set of elements which appear in some other set

- Examples:
 - The words used in a document or sentence
 - All the words of a language
 - The characters used in a word
 - The English alphabet is the ‘vocabulary’ of symbols used in writing the language

Tallying

Tallies:

The count (number of occurrences) for a each distinct element of a non-distinct set

- Each element becomes associated with its count
- Tallying creates a distinct set of tuples (a language model) from a non-distinct set (a corpus)
- Tallying is probably the most commonly used programming pattern in corpus linguistics

Tally Example

"a man gave sandy a pomelo"

```
(a, 2)
(man, 1)
(gave, 1)
(sandy, 1)
(pomelo, 1)
```

A tally is a count. We can divide each tally by the number of instances to obtain *frequencies*.

```
(a, .33)
(man, .16)
(gave, .16)
(sandy, .16)
(pomelo, .16)
```

Combining Counts

- Fundamental counting principle:

The cross product of the sets

$\{m_0 \dots m_{cc_{mm}}\}$ and $\{n_0 \dots n_{cc_{nn}}\}$

has $cc_{mm} \times cc_{nn}$ members

- That is, there are $m \times n$ ways to combine exactly one element each from sets of size m and n .
- Combine independent counts by multiplying

Combining Counts

- It follows that if we choose m times from the same set of n elements without depletion (or with replacement), there are n^m possible results. (This includes all possible orderings)

Example:

Simplistically, how many four-letter words can be formed with the English alphabet?

$$26^4 = 456,976$$

Counting Sets: Combinatorics

- The basics of counting and arranging sequences:

Permutations:

all possible orderings of n elements

Combinations:

all (unordered) sets of k elements that can be selected from n elements

Variations:

all orderings of k elements that can be selected from n elements.

Permutations

{ o, r, a, n, g, e }

6 choices	<div>a</div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	{ o, r, n, g, e }
5 choices	<div>a</div>	<div>e</div>	<div></div>	<div></div>	<div></div>	<div></div>	{ o, r, n, g }
4 choices	<div>a</div>	<div>e</div>	<div>g</div>	<div></div>	<div></div>	<div></div>	{ o, r, n }
3 choices	<div>a</div>	<div>e</div>	<div>g</div>	<div>n</div>	<div></div>	<div></div>	{ o, r }
2 choices	<div>a</div>	<div>e</div>	<div>g</div>	<div>n</div>	<div>o</div>	<div></div>	{ r }
1 choice	<div>a</div>	<div>e</div>	<div>g</div>	<div>n</div>	<div>o</div>	<div>r</div>	{ }

$$6 \times 5 \times 4 \times 3 \times 2 \times 1 =$$
$$6! = 720$$



This is the factorial function,
denoted by $n!$

Permutations

- The previous example, ‘orange’ had no repeated letters in the input.
- Permutation only rearranges elements, so it can never “generate” repetition (like we will see with combinations).
- But if the input contains repetitions of some value(s), we may want to eliminate duplicate outputs.
- To do this, we simply divide $n!$ by the number of those combinations that are indistinguishable from each other
- Applying the fundamental counting principle to combine each set of repeated values $\{r_{i,0}, r_{i,1}, \dots, r_{i,m}\}$ in the input, this denominator is given by:

$$\prod_i m_{ii}!$$

Permutations with Repeated Input Values

Example:

mississippi

11 letters

repeated group { i, i, i, i } : cardinality 4

repeated group { s, s, s, s } : cardinality 4

repeated group { p, p } : cardinality 2

$$\frac{11!}{4! \times 4! \times 2!} = 34,650$$



Some sources consider counting the permutations of a subset of a set's elements to be a type of permutation, but I treat this separately—see 'Variations'

Combinations

- Also called k -combinations
- How many unordered sets of size k can be formed from a set of cardinality n ?
- We've already considered the case of not depleting (i.e., replacing) when ordering of the output matters:

n^k (a.k.a. variations, with repetition)



And as for unordered sets of k from n with replacement, see Assignment 1 - extra credit

- If depleting, we can derive this answer from the fundamental counting principle

Combinations

Example: We have seven documents and we need to find the group of three documents that is most divergent from the corpus. How many groups of 3 will we have to test?

1st choice: 7 documents to choose from

2nd choice: 6 remaining documents

3rd choice: 5 remaining documents.

$$7 \times 6 \times 5 = 210 \text{ sets.}$$

But ordering doesn't matter. Since each of these sets has 3! orderings, we divide by $3! = 6$, thus counting each set just once.

$$210 \div 6 = 35$$

Combinations

- How can we generalize $n(n-1) \dots (n-k+1)$

Let's use factorial:

$$\frac{n(n-1) \dots (n-k+1)(n-k) \dots k \dots (2)(1)}{(n-k) \dots k \dots (2)(1)} = \frac{n!}{(n-k)!}$$

Don't forget to cancel out the permutations, since ordering doesn't matter:

$$\binom{n}{k} = \frac{n!}{(n-k)! k!}$$



This is called the binomial coefficient, or choose function.

Combinations

- Binomial coefficient
- “n choose k”

upper index \rightarrow

lower index \rightarrow

$$\binom{n}{k} = \frac{n!}{(n - k)! k!}$$

- Gives the number of subsets of size k that can be formed from a set of size n

Permutation + Combination

- Permutation: how many different orderings?
- Combination: how many different subsets?
- Variations: how many different ordered subsets?

Repetition v. Replacement

{ set }	unordered, distinct (no repetition)
{ multiset, bag }	unordered, repetition permitted
(tuple)	ordered, repetition permitted

- For combinations (not permutations) we can allow repetition in the *output*

Do we want the combinations of “{ a b c } choose 2” to include { a a }—in which case there are 6 output sets—or not—in which case there are only 3: { a b }, { a, c }, { b, c }

Repetition: Distinctness within a collection

- If the *input* has repeated values, we must decide whether we want to count repeated *output* tuples that might result:

The $3! = 6$ **permutations** of $\{a a b\}$ consists of two instances each of $(a a b)$, $(a b a)$, and $(b a a)$.

By the nature of permutations, it's clear that usually we're not interested in the duplicate tuples.

However the **combinations** of “ $\{a a b\}$ choose 2” includes two instances of $\{a a\}$

...and depending on the application, we may or may not be interested in the extra (e.g. $\{aa\}$ is twice as likely as $\{ab\}$)

Variations

- Variations are permutations of combinations: ordered subsets of size k chosen from a set of n .
- With repetition in the output:

$$n^k$$

- *Without repetition in the output:*

Same as the way we derived the choose function, but we don't need to cancel out the repeated outputs

$$\frac{n!}{(n - k)!}$$



Permutations allowing repetition in the *output*

- In permutations, there is no sense of an element being “chosen” more often. We include every arrangement of input items without examining their values
- Permutations which allow repetitions in the output are the same as variations where the lower index equals the upper index.
 - Everything is fine when the input set is distinct
 - There are n^k output tuples, and they’re all different
 - When there is repetition in the input, we have another choice to make, namely:
 - should duplicate output tuples be counted?
- These points are shown on the next two slides

Permutations v. Variations

		output type	
		permutations	variations with repetition where lower index = input cardinality
input	distinct { a, b, c }	$\{ a, b, c \}, \{ a, c, b \},$ $\{ b, a, c \}, \dots \{ c, b, a \}$ $n!$	(a, a, a) (a, a, b) (a, a, c) \dots (c, c, c) n^n
	has repetition (a, a, b)	$(a a b), (a b a)$ $(a a b), (a b a)$ $(a b a), (b a a)$ $(a b a), (b a a)$ $\frac{n!}{\prod_i (m_i!)}$	(a, a, a) (a, a, b) (a, b, a) \dots (b, b, b) $(\text{\#distinct elements})^n$

Duplicate output tuples

- Input (has repetition)

(a, a, b)

Variations (lower index=3) allowing repetition in the output: 27 of them

(a a a) (a a a) (a a b) (a a a) (a a a) (a a b) (a b a)
(a b a) (a b b) (a a a) (a a a) (a a b) (a a a) (a a a)
(a a b) (a b a) (a b a) (a b b) (b a a) (b a a) (b a b)
(b a a) (b a a) (b a b) (b b a) (b b a) (b b b)

There are only 8 different output tuples

Combinatorics Summary

$\{a b c\}$

- Permutation: how many different orderings?

$(a b c)(a c b)(b a c)(b c a)(c a b)(c b a)$ $n!$

- Combination: how many different subsets (i.e. of 2)?

$\{a b\}\{a c\}\{b c\}$

allowing repetition in the output

$\{a a\}\{a b\}\{a c\}\{b b\}\{b c\}\{c c\}$



Assignment 1 extra credit

$\begin{matrix} ? & n \\ & \diagdown \diagup \\ & k \end{matrix}$

- Variations: how many different ordered subsets (i.e. of 2)?

$(a b)(a c)(b a)(b c)(c a)(c b)$

allowing repetition in the output

$(a a)(a b)(a c)(b a)(b b)(b c)(c a)(c b)(c c)$

$\frac{n!}{(n - k)!}$

n^k

Unix

- Bell Labs, 1969: Thompson, Ritchie, et al.
- Simple model: a UI-less ‘kernel’ provides process, device, and memory management
- Command line shells provide interactive interaction, if required:
 - sh, csh, ksh, bash
- Historical progression of implementations
 - System V, BSD, POSIX, Linux, Mac OS X
- X-Windows: a graphical interface to the kernel
- KDE, Gnome: graphical desktops

Shell commands: files

\$ ls	list files in the current directory
\$ cat	show the contents of a file
\$ cp	copy a file
\$ mv	move or rename a file
\$ rm	delete a file

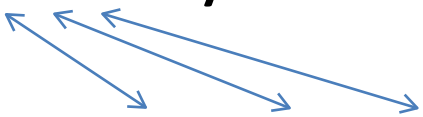
These are just programs, really.

Since filenames are case sensitive, so are these command names.



File permissions

\$ chmod +x myfile
makes myfile executable

\$ chmod 774 myfile



binary: 111111100

set r w x r w x r - -
   
for owner, group, and others

Shell commands: directories

\$ cd change working directory

\$ mkdir make a new directory

\$ rmdir remove a directory

/ separates directory paths

. refers to the current directory

.. refers to the parent directory

Unix console control

ctrl-D	end the program, end of stream this is ctrl-Z on DOS
ctrl-S	XOFF, pause the stream (if supported) beware appearance of hang
ctrl-Q	XON, resume the stream (if supported)
ctrl-C	attempt to interrupt the program
ctrl-L	FF, form feed, clear the screen
ctrl-I	HT, horizontal tab
ctrl-M	CR, carriage return (see next slide)
ctrl-J	LF, line feed (see next slide)

Text file line endings

- Different systems use different conventions for line endings in text files (i.e. corpora)
- Since files are migrated between systems, we will always need to handle these differences correctly

System	Line ending convention	ASCII	Unicode	C
Unix	LF	0A	000A	\n
DOS/Windows	CR LF	0D 0A	000D 000A	\r\n
Macintosh (Pre-OS X)	CR	0D	000D	\r

Editors

- ed
 - command line editor, ~1971
 - first implementation of regular expressions
- vi, vim
- emacs
- nano
- pico
- other solutions: local editor with ssh script

Shell scripts

myprog.sh

'shebang'



```
#!/bin/sh  
  
# the hash mark indicates a comment  
line ls /
```

to run this program:

```
$ ./myprog.sh
```

notice the mention of the current directory

Standard I/O handles

Unix uses the concept of ‘streams’ of characters.

0	Standard Input	stdin
1	Standard Output	stdout
2	Standard Error	stderr

Unix shell: pipes and redirection

\$ more	show output one screen at a time
\$ tail	show the last 10 lines of a file
\$ cat foo >bar	redirect contents of 'foo' to stdout
\$ cat foo 1>bar	same thing

\$./myprog <foo >bar 2>errout

execute 'myprog,' pass the contents of 'foo' in as standard input, capture standard output to 'bar,' and capture standard error to 'errout'

More redirection

- > redirect output to a new file
- < redirect input from a file
- >> append output to a new or existing file
- &> redirect stdout and stderr to a new file
- | pipe stdout to the next program as stdin

```
$ ls -l | sort
```

Text processing utilites

- `wc` word count
 - arguments `-l`, `-w`, `-c` count lines, words, characters
- `sort` general purpose ASCII or ordinal sort
 - It's not encoding-aware which renders it for serious linguistic use
- `tr` translate (substitute character ranges)
- `grep` search for matching patterns
- `sed` stream editor
- `uniq` remove duplicate lines (from sorted files)
- `diff` compare text files

Working with data

- Basic definitions:

bit: a single memory cell that can have the value zero or one

byte: a fixed group of 8 (eight) ordered, distinct bits
therefore, a byte has a value between 0 and $(2^8 - 1 = 255)$

MSB: the most-significant-bit in a byte
LSB: the least-significant-bit in a byte

KB: one kilobyte: $2^{10} = 1024$ bytes
MB: one megabyte: $2^{20} = 1,048,576$ bytes
GB: one gigabyte: $2^{30} = 1,073,741,824$ bytes
4 GB: four gigabytes: $2^{32} = 4,294,967,296$ bytes

8-bit character encodings

- 8-bit: $2^8 = 256$ possible characters
 - characters 0-127: usually ASCII, fairly standardized
 - characters 127-255: a free-for-all
- Hundreds of different systems for assigning various characters to the 256 available positions
- Each of these is a **character encoding**



A (partial) list of some 8-bit character encodings

IBM037	IBM EBCDIC (US-Canada)	windows-1257	Baltic (Windows)	IBM871	IBM EBCDIC (Icelandic)
IBM437	OEM United States	windows-1258	Vietnamese (Windows)	IBM880	IBM EBCDIC (Cyrillic Russian)
IBM500	IBM EBCDIC (International)	Johab	Korean (Johab)	IBM905	IBM EBCDIC (Turkish)
ASMO-708	Arabic (ASMO 708)	macintosh	Western European (Mac)	IBM00924	IBM Latin-1
DOS-720	Arabic (DOS)	x-mac-japanese	Japanese (Mac)	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
ibm737	Greek (DOS)	x-mac-chinesetrad	Chinese Traditional (Mac)	x-cp20936	Chinese Simplified (GB2312-80)
ibm775	Baltic (DOS)	x-mac-korean	Korean (Mac)	x-cp20949	Korean Wansung
ibm850	Western European (DOS)	x-mac-arabic	Arabic (Mac)	cp1025	IBM EBCDIC (Cyrillic Serbian-Bulgarian)
ibm852	Central European (DOS)	x-mac-hebrew	Hebrew (Mac)	koi8-u	Cyrillic (KOI8-U)
IBM855	OEM Cyrillic	x-mac-greek	Greek (Mac)	iso-8859-1	Western European (ISO)
ibm857	Turkish (DOS)	x-mac-cyrillic	Cyrillic (Mac)	iso-8859-2	Central European (ISO)
IBM00858	OEM Multilingual Latin I	x-mac-chinesesimp	Chinese Simplified (Mac)	iso-8859-3	Latin 3 (ISO)
IBM860	Portuguese (DOS)	x-mac-romanian	Romanian (Mac)	iso-8859-4	Baltic (ISO)
ibm861	Icelandic (DOS)	x-mac-ukrainian	Ukrainian (Mac)	iso-8859-5	Cyrillic (ISO)
DOS-862	Hebrew (DOS)	x-mac-thai	Thai (Mac)	iso-8859-6	Arabic (ISO)
IBM863	French Canadian (DOS)	x-mac-ce	Central European (Mac)	iso-8859-7	Greek (ISO)
IBM864	Arabic (864)	x-mac-icelandic	Icelandic (Mac)	iso-8859-8	Hebrew (ISO-Visual)
IBM865	Nordic (DOS)	x-mac-turkish	Turkish (Mac)	iso-8859-9	Turkish (ISO)
cp866	Cyrillic (DOS)	x-mac-croatian	Croatian (Mac)	iso-8859-13	Estonian (ISO)
ibm869	Greek, Modern (DOS)	x-Chinese-CNS	Chinese Traditional (CNS)	iso-8859-15	Latin 9 (ISO)
IBM870	IBM EBCDIC (Multilingual Latin-2)	x-cp20001	TCA Taiwan	x-Europa	Europa
windows-874	Thai (Windows)	x-Chinese-Eten	Chinese Traditional (Eten)	iso-8859-8-i	Hebrew (ISO-Logical)
cp875	IBM EBCDIC (Greek Modern)	x-cp20003	IBM5550 Taiwan	iso-2022-jp	Japanese (JIS)
shift_jis	Japanese (Shift-JIS)	x-cp20004	TeleText Taiwan	csISO2022JP	Japanese (JIS-Allow 1 byte Kana)
gb2312	Chinese Simplified (GB2312)	x-cp20005	Wang Taiwan	iso-2022-jp	Japanese (JIS-Allow 1 byte Kana - SO/SI)
ks_c_5601-1987	Korean	x-IA5	Western European (IA5)	iso-2022-kr	Korean (ISO)
big5	Chinese Traditional (Big5)	x-IA5-German	German (IA5)	x-cp50227	Chinese Simplified (ISO-2022)
IBM1026	IBM EBCDIC (Turkish Latin-5)	x-IA5-Swedish	Swedish (IA5)	euc-jp	Japanese (EUC)
IBM01047	IBM Latin-1	x-IA5-Norwegian	Norwegian (IA5)	EUC-CN	Chinese Simplified (EUC)
IBM01140	IBM EBCDIC (US-Canada-Euro)	us-ascii	US-ASCII	euc-kr	Korean (EUC)
IBM01141	IBM EBCDIC (Germany-Euro)	x-cp20261	T.61	hz-gb-2312	Chinese Simplified (HZ)
IBM01142	IBM EBCDIC (Denmark-Norway-Euro)	x-cp20269	ISO-6937	GB18030	Chinese Simplified (GB18030)
IBM01143	IBM EBCDIC (Finland-Sweden-Euro)	IBM273	IBM EBCDIC (Germany)	x-iscii-de	ISCII Devanagari
IBM01144	IBM EBCDIC (Italy-Euro)	IBM277	IBM EBCDIC (Denmark-Norway)	x-iscii-be	ISCII Bengali
IBM01145	IBM EBCDIC (Spain-Euro)	IBM278	IBM EBCDIC (Finland-Sweden)	x-iscii-ta	ISCII Tamil
IBM01146	IBM EBCDIC (UK-Euro)	IBM280	IBM EBCDIC (Italy)	x-iscii-te	ISCII Telugu
IBM01147	IBM EBCDIC (France-Euro)	IBM284	IBM EBCDIC (Spain)	x-iscii-as	ISCII Assamese
IBM01148	IBM EBCDIC (International-Euro)	IBM285	IBM EBCDIC (UK)	x-iscii-or	ISCII Oriya
IBM01149	IBM EBCDIC (Icelandic-Euro)	IBM290	IBM EBCDIC (Japanese katakana)	x-iscii-ka	ISCII Kannada
windows-1250	Central European (Windows)	IBM297	IBM EBCDIC (France)	x-iscii-ma	ISCII Malayalam
windows-1251	Cyrillic (Windows)	IBM420	IBM EBCDIC (Arabic)	x-iscii-gu	ISCII Gujarati
Windows-1252	Western European (Windows)	IBM423	IBM EBCDIC (Greek)	x-i scii-pa	ISCII Punjabi
windows-1253	Greek (Windows)	IBM424	IBM EBCDIC (Hebrew)		
windows-1254	Turkish (Windows)	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean Extended)		
windows-1255	Hebrew (Windows)	IBM-Thai	IBM EBCDIC (Thai)		
windows-1256	Arabic (Windows)	koi8-r	Cyrillic (KOI8-R)		

8-bit encodings

- Great for 1968
 - why?
- Not so great for 2015
 - why?



File encodings

- A file with 8-bit characters generally has no internal provision for identifying which encoding it uses
- This information must be specified in some kind of metadata
 - out-of-band
 - the file name extension?
 - perhaps you just happen to know
 - somebody told you
 - you guess (or use a probabilistic model) by inspecting the contents

Tim Baldwin and Marco Lui. 2010. Language Identification: The Long and the Short of the Matter. *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL*. ACL

- in-band
 - i.e. HTML:
<meta http-equiv="Content-Type" content="text/html; charset=TIS-620" />

Unicode

- Unicode uses 16-bits to store each character
 - $2^{16} = 65535$ possible characters
 - Major languages are well represented
 - Standard assignments: no conflicting characters
 - Combining characters for accents, ligatures
 - Unicode layout engines are more intelligent than just a monospace console
 - Compatible: characters 0-127 are the same as ASCII
- A single Unicode character is called a **code point**
- Unicode text is a stream of code points

UTF-8

- Unicode is nice, but hey, my documents are 99% English. Why do they have to be twice as big?
- 8-bit Unicode Transformation Format (UTF-8) uses a variable number of bytes to encode Unicode characters
- In fact, if you only use ASCII, the UTF-8 stream looks like an 8-bit ASCII stream
- 1, 2, 3, or 4 bytes per character are used
 - this means that some Unicode streams get *larger* using UTF-8
 - This will probably be true for alphabets/languages other than:
Extended Latin alphabet, Romance languages, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Tāna

How does UTF-8 work?

Unicode range		Encoded bytes	Example
Hex	Binary		
U+0000 to U+007F	00000000 to 01111111	0xxxxxxx	'\$' U+0024
			= 00100100
			→ 00100100
			→ 0x24
U+0080 to U+07FF	00000000 10000000 to 00000111 11111111	110yyyxx	'ç' U+00A2
		10xxxxxx	= 00000000 10100010
			→ 11000010 10100010
			→ 0xC2 0xA2
U+0800 to U+FFFF	00001000 00000000 to 11111111 11111111	1110yyyy	'€' U+20AC
		10yyyyxx	= 00100000 10101100
		10xxxxxx	→ 11100010 10000010 10101100
			→ 0xE2 0x82 0xAC
U+010000 to U+10FFFF	00000001 00000000 00000000 to 00010000 11111111 11111111	11110zzz	'𐐦' U+024B62
		10zzyyyy	= 00000010 01001011 01100010
		10yyyyxx	→ 11110000 10100100 10101101 10100010
		10xxxxxx	→ 0xF0 0xA4 0xAD 0xA2

Using HTML <meta> tag to specify encoding

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

This is nice, but it you still have to:

- Save the file using the specified encoding
 - This requires using an editor that is capable of doing so
- Configure the web server to send a matching HTTP header
 - This is an out-of-band mechanism for specifying the content encoding of every single web page

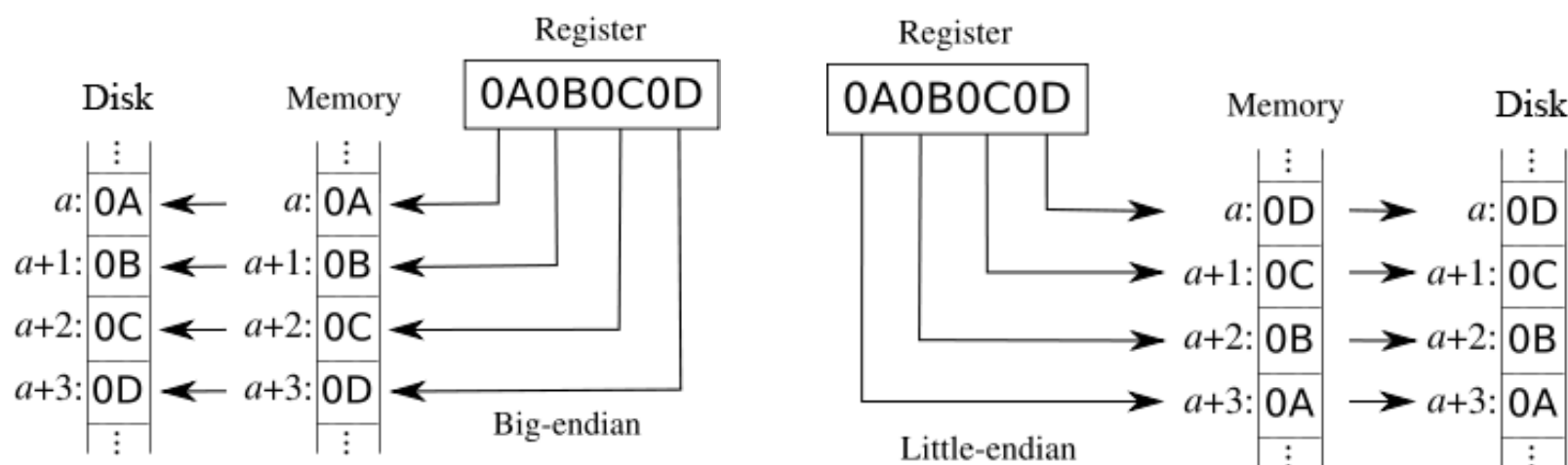
```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-
Hat/Linux) Last-Modified: Wed, 08 Jan 2003
23:11:55 GMT Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges:
bytes Content-
Length: 438
Connection: close
```

Endianness

- The layout of 16-bit or 32-bit values in memory is microprocessor dependent
 - nowadays, this is not an issue for bytes
- If a disk file is just a copy of a memory image, then this difference can persist in the file
 - **big endian:**
most-significant-byte ... least-significant-byte
 - **little endian:**
least-significant-byte ... most-significant-byte

Why this matters to computational linguistics

- Since unicode uses 16-bits per character, endianness sometimes matters



- UTF-8 is defined a stream of bytes, however, so it is not affected by this issue

Byte-order Mark (BOM)

- Solving the ‘endian-ness’ problem for Unicode files
- But also used as in-band means for distinguishing Unicode file formats UTF-8 and UTF-16
- Infrequently used, but important for computational linguists to be aware of
- Examine the first few bytes of a text file for the BOM

Encoding	Representation (hexadecimal)	Representation (decimal)	Representation (ISO-8859-1)
UTF-8	EF BB BF	239 187 191	ï»¿
UTF-16 (BE)	FE FF	254 255	þÿ
UTF-16 (LE)	FF FE	255 254	ÿþ



Firefox browser does not like to see a BOM at the top of an HTML file

Programming language support for encodings

```
String s = "สวัสดีครับ";           // C# strings are always unicode
int i = s.Length;                   // number of code points: 10
Char ch = s[0];                     // always a 16-bit character.
                                    // In this case the value is 3626 (U+0E2A)

Byte[] bytes = Encoding.GetEncoding("TIS-620").GetBytes(s);
i = bytes.Length;                   // number of bytes: 10
byte b = bytes[0];                  // a byte. In this case, the value is 202 (\xCA)

bytes = Encoding.UTF8.GetBytes(s);
i = bytes.Length;                   // number of bytes: 30

s = Encoding.UTF8.GetString(bytes); // back to the original string
```

Regular Expressions

- Regular expressions: a syntax for matching patterns in text
 - <1950s Automata theory
 - 1950s Stephen Kleene
 - 1960s SNOBOL
 - 1970s Ken Thompson – QED
 - Unix – ed
 - grep
 - Perl
 - etc...

Basic RegEx

- `^` matches the start of a line
- `$` matches the end of a line
- `.` matches any one character (except newline)
- `[xyz]` matches any one character from the set
- `[^pdq]` matches any one character not in the set
- `|` accepts either its left or its right side
- `\` escape to specify special characters
- anything else: must match exactly

More RegEx

- * accepts zero or more of the preceding element
this is the canonical 'greedy' operator
- ? accepts zero or one of the preceding element(s)
- + accepts one or more of the preceding element(s)
- {*n*} accepts *n* of the preceding element(s)
- {*n*,} accepts *n* or more of the preceding element(s)
- {*n*,*m*} accepts *n* to *m* of the preceding element(s)

- (*pattern*) defines a capture group which can be referred to later via \1

RegEx Examples

- Find the English stops followed by a liquid
`grep [PDKBDGpdkbdg][lr]`
- Find any two vowels together
`grep [aeiou][aeiou]`
- Find the same letter, repeated
`egrep '([a-z])\1'`
- Lines where sentences end with 'to'
`egrep '(|^)to.'`

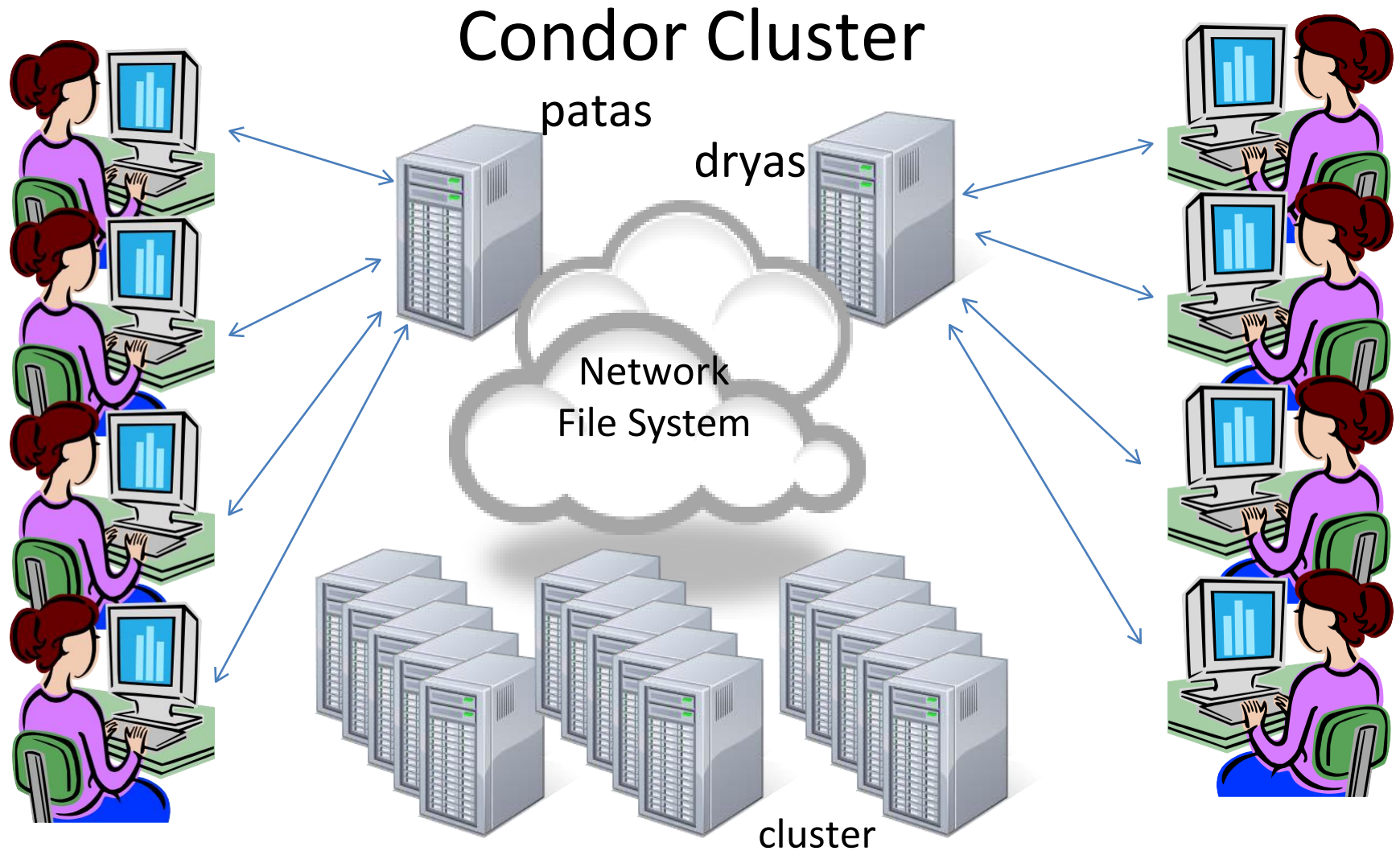
Primitive tokenization

```
$ cat moby_dick.html |           # echo the text
tr [:upper:] [:lower:] |       # convert to lower case
tr ' ' '\n' |                  # put each word on a line
grep -v ^$ |                   # get rid of blank lines
grep -v '<' |                   # get rid of HTML tags
grep -o "[a-z']*" |            # only want letters and '
sort |                         # sort the words
uniq |                         # find the vocabulary
wc -l                          # count them
```

3956



If you are going to attempt to do project 1 using only unix shell utilities, you might need something like this



Condor

```
$ condor_submit myjob.cmd
```

```
universe          = vanilla
executable        = /usr/bin/python
getenv            = true
input             = myinput.in
output            = myoutput.out
error             = myerror.err
log               = /tmp/gslayden/mylogfile.log
arguments         = "myprogram.py -x"
transfer_executable = false
queue
```

The system will send you email when your job is complete.

Using variables in Condor files

flexible.job

```
file_ext      = $(depth)_$(gain)
universe      = vanilla
executable    = /opt/mono/bin/mono
getenv        = true
output        = acc_file.$(file_ext)
error         = q4.err
log           = /tmp/gslayden/q4.log
arguments     = "myprog.exe model_file.$(file_ext) sys_file.$(file_ext)"
transfer_executable = false
queue
```

```
$ condor_submit -append "depth=20" -append "gain=4" flexible.job
```

One step

```
$ condor-exec myprogram.py
```

Thanks to Bill McNeil for writing this handy script.

But for Project 1 you must write your own

Project 1

- Due August 4, 11:45 p.m.
- Counting syntactic constituents in a corpus
- You may use regular expressions for this
 - (not a requirement if you prefer procedural code)
- Using Linux
- Using the Condor system
- Packaging and submitting assignments

Next Time

- Events, Probability, Distributions