Lecture 3

July 28, 2016

Probability

● Reminder: start the recording

Announcements

- Probability textbook:
  - Charles Grinstead and J. Laurie Snell *Introduction to Probability* - online on course website
- Resources on course website
  - Unicode, UTF-8, BOM
  - Condor quick-start
  - Probability for Linguists
  - Probability summary

Project 1

- Due at 11:45 p.m. next Thursday
- Condor and RegEx covered at the beginning of this lecture
- Unix or system issues?
- Questions?

Assignment 2

- Now posted, due August 9th
- Probability (to be covered soon)

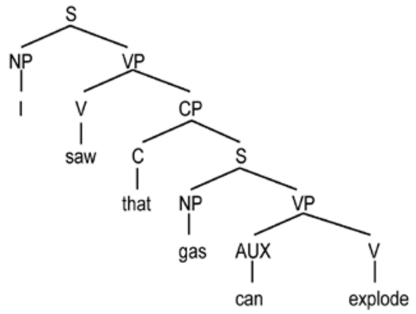
Writing assignment

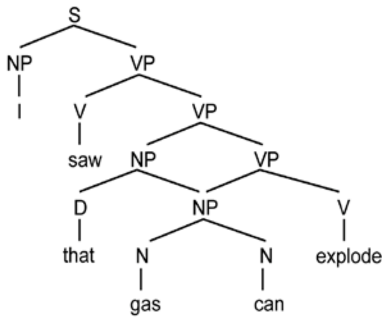
- Due September 6th , 2016

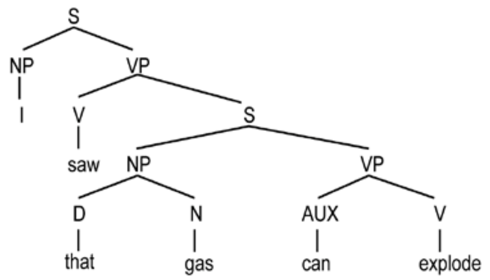
  http://courses.washington.edu/ling473/writing-assignment.html
- Short Critical review of a paper from the computational linguistics literature
- Formatted according to ACL guidelines
  – http://acl2016.org/files/acl2016.zip
  – For MS Word, see http://acl2015.org/call_for_papers.html
- Any published journal or peer-reviewed paper on a comp. ling. topic is acceptable

Assignment 1

1. Thank you for your essays. Full credit
2. *I saw that gas can explode.*
   - I realized that gas (in general) is able to explode.
   - I (literally) saw that (particular) gas container explode.
   - I realized (that) that (particular) gas is able to explode.
   - etc…







3. All possible 6-letter words (26^6)

subtract words with all consonants (21^6)

subtract words with all vowels (5^6)

$26^6 - 21^6 - 5^6 = 223{,}134{,}030$

4. Assuming that we consider identical characters to be indistinguishable in the output:

(萄 萄 萄 萄 橙 橙 苹 梨 蕉)

repeated groups: 4, 2, 1, 1, 1

$$\frac{9!}{4! \times 2! \times 1! \times 1! \times 1!} = 7{,}560$$

5. How many pairwise comparisons are possible between documents on the same topic?

$$\binom{7}{2} + \binom{9}{2} + \binom{3}{2} = 60$$

How many pairwise comparisons are possible between documents on different topics?

$$(7 \times 9) + (7 \times 3) + (9 \times 3) = 111$$

6. Extra Credit

Write an expression that gives the number of unordered sets of *k* items that can be formed from a set of *n* distinct items while allowing repetition in the output set.

example: { a, b, c, d } choose 3 (unordered), but allowing repetition in the output:

{ a a a }, { a a b }, { a a c }, { a a d },
{ a b b }, { a b c }, { a b d }, { a c c },
{ a c d }, { a d d }, { b b b }, { b b c },
{ b b d }, { b c c }, { b c d }, { b d d },
{ c c c }, { c c d }, { c d d }, { d d d }

We ensure that order doesn't matter (i.e. we're not including duplicate tuples) by enumerating them using the convention of displaying the letters in sorted order. So we know the answer should be 20.

Divide into groups and count them (remember: $n = 4$, $k = 3$):

a-group	{ a a a }{ a a b }{ a a c }{ a a d }{ a b b }
	{ a b c }{ a b d }{ a c c }{ a c d }{ a d d }
	$\binom{4}{2} + 4 = 10$

b-group	{ b b b }{ b b c }{ b b d }{ b c c }{ b c d }{ b d d } $\binom{3}{2} + 3 = 6$
c-group	{ c c c }{ c c d }{ c d d } $\binom{2}{2} + 2 = 3$
d-group	{ d d d } $\binom{1}{2} + 1 = 1$

$$\sum_{i}^{n} \binom{i}{k-1} + i = \binom{n+k-1}{k} = \left(\!\binom{n}{k}\!\right)$$

This is called the multiset coefficient

see http://en.wikipedia.org/wiki/Multiset

and http://en.wikipedia.org/wiki/Stars_and_bars_(probability)

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 } $\binom{n}{k}$

  allowing repetition in the output

  { a a } { a b } { a c } { b b } { b c } { c c } Assignment 1 extra credit

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

  (a b) (a c) (b a) (b c) (c a) (c b) $\frac{n!}{(n-k)!}$

  allowing repetition in the output

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

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-iscii-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 2012
  - 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" />
```

Encoding issues to keep straight

1. Your text editor
   - Can your editor create, display and save a file with wide literals (typ. UTF-8)?
2. Language support for wide characters
   - Does your compiler support wide literals in source code files?
   - If so, does it need/expect/reject a BOM? Or do you perhaps need to specify the source file's encoding on the compiler command line?
   - If not, does it support wide characters through via ASCII escape?
3. Utility programs:
   - Does your FTP program correctly understand UTF-8 files when it tries to convert line-endings from Windows to Unix? (try binary mode)
4. At runtime:
   - Does your environment have functions for reading unicode files?
   - Which input file are you trying to read? { UTF-8, UTF-16LE, TIS-620 }
   - Does your environment have functions for writingunicode files?
   - What output encoding are you trying to write? { UTF-8, UTF-16LE, TIS-620 }

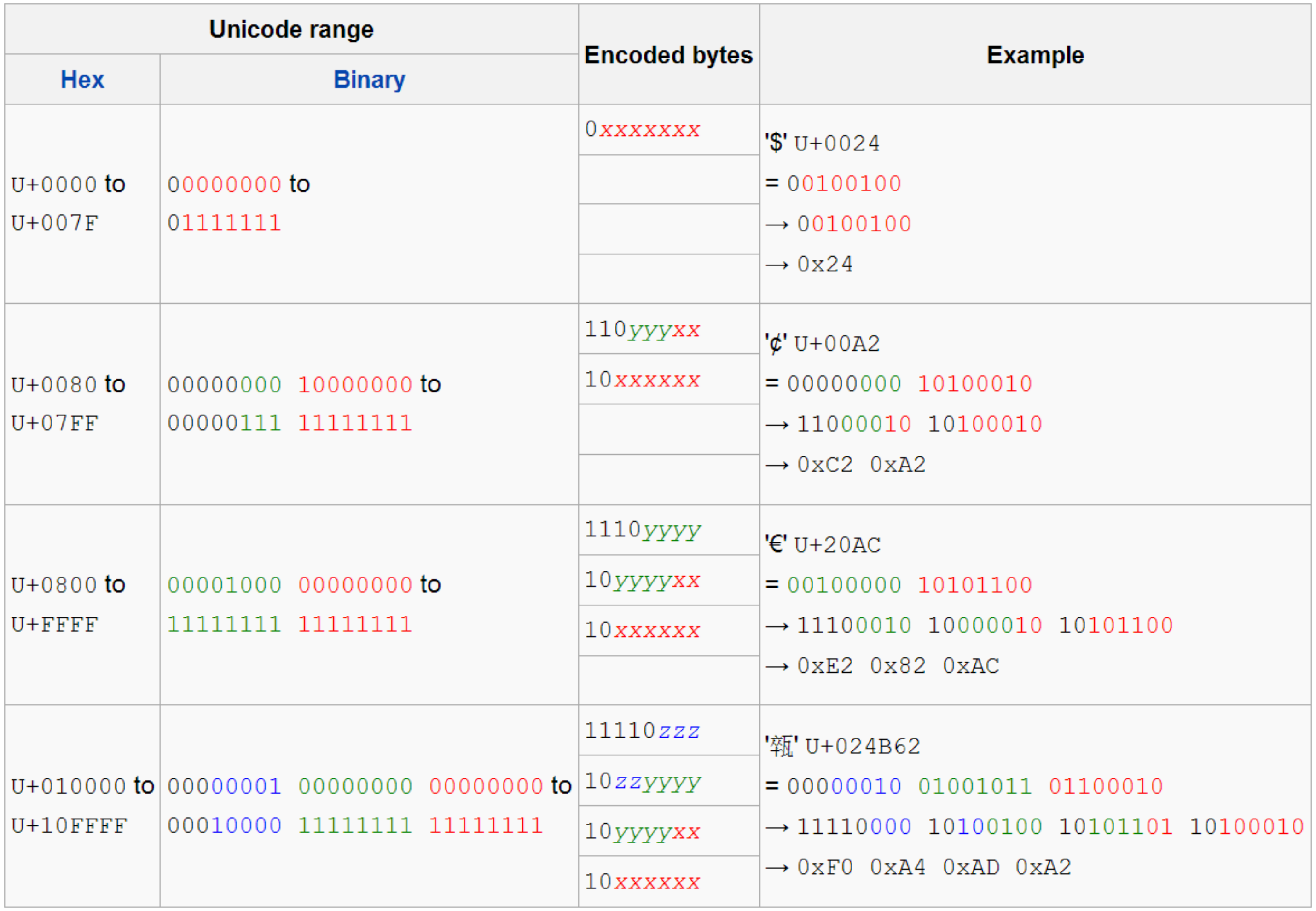
Unicode

- 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 10xxxxxx	'¢' U+00A2 = 00000000 10100010 → 11000010 10100010 → 0xC2 0xA2
U+0800 to U+FFFF	00001000 00000000 to 11111111 11111111	1110yyyy 10yyyyxx 10xxxxxx	'€' U+20AC = 00100000 10101100 → 11100010 10000010 10101100 → 0xE2 0x82 0xAC
U+010000 to U+10FFFF	00000001 00000000 00000000 to 00010000 11111111 11111111	11110zzz 10zzyyyy 10yyyyxx 10xxxxxx	'𤭢' U+024B62 = 00000010 01001011 01100010 → 11110000 10100100 10101101 10100010 → 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
Content-Type: text/html; charset=UTF-8
```

Endianness

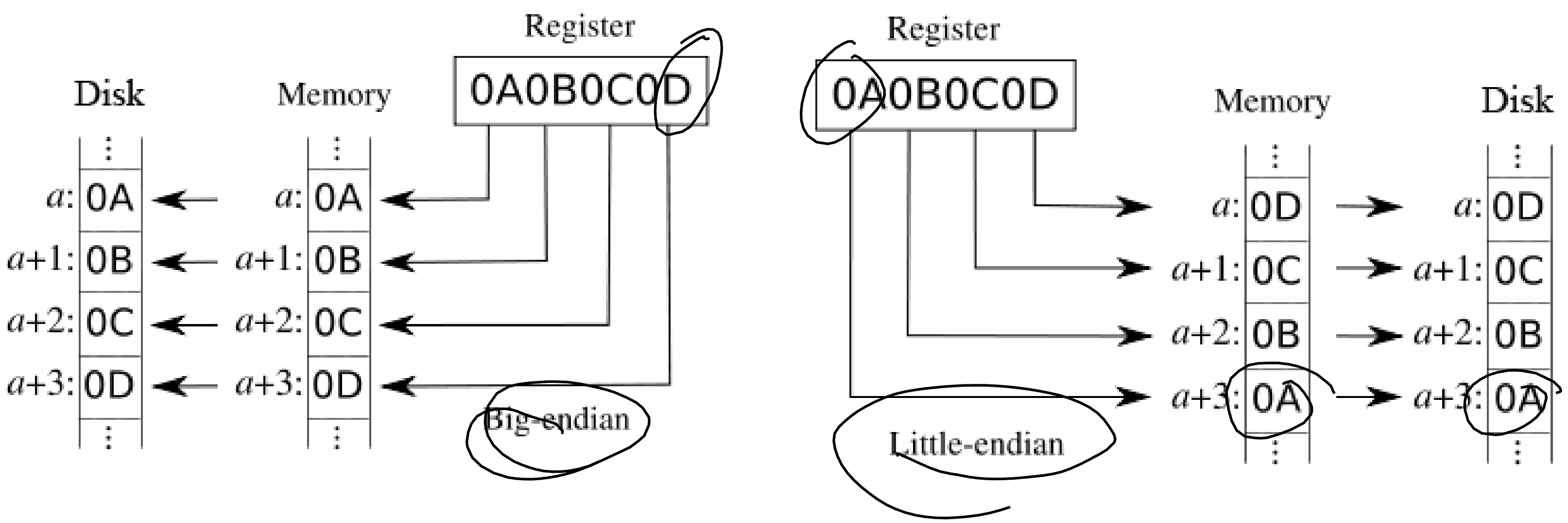
- The layout of 16-bit or 32-bit values in memory is microprocessor dependent
  - nowadays, this is not an issue for bytes
- If a disk file is just a copy of a memory image, then this difference can persist in the file
  - big endian:

    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

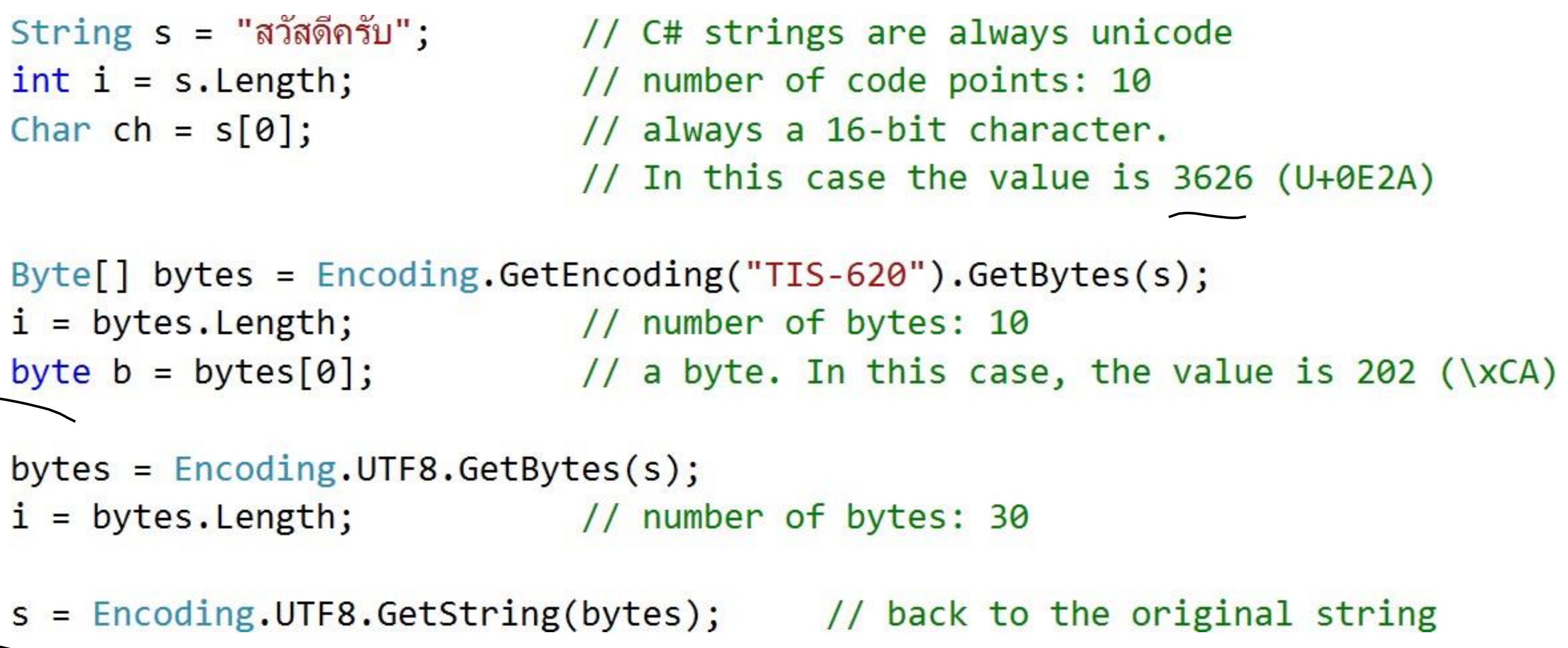
Byte-order Mark (BOM)

- 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\.'
  ```

Unix pipe utilities

- more, less (paginate for console screen)
- tr (transpose, substitute)
- uniq (unique elements, make sequence distinct)
- sort (sort)
- wc (word and line count info)
- others?

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.$(file_ext)
error               = q4.err
log                 = /tmp/your-uw-netid/q4.log
arguments           = “myprog.exe model_file.$(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

Probability

- So far, we have considered counting and arranging sets of items (entities, elements)
- If we consider the event of selecting an element from a set, we enter the world of probability
- This event could also be called an observation or a trial
- The set becomes known as the sample space and—by definition—it is assigned a probability mass of 1.0

Sample Spaces

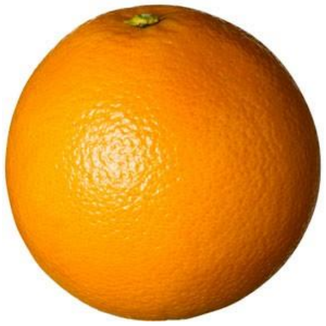
- Ω (omega) is often used to represent the sample space
- Sample spaces can be discrete or continuous

    discrete:

  Ω = (apple, banana, banana, orange)

    continuous:

  Ω = { *the mass of an orange* }



- P(Ω) = 1 (all possible events are accounted for)

most applications in computational linguistics involve discrete probabilities

Outcomes

- Often, events are often notated with a italic capital letter corresponding to a single outcome (a lower-case letter)

  Ω = (a a b c)

  A is the event of selecting 'a' from Ω

  B is the event of selecting 'b' from Ω

  C is the event of selecting 'c' from Ω

- A^C denotes the complement of event *A*
  - An event *A* partitions the sample space into *A* and A^C
- An event can also be any subset of Ω
  - All individual outcomes are events, but events can also be combinations of individual outcomes

  e.g. Q is the event of selecting 'b' and 'a' from Ω, in that order

Rolling 2 dice

- The single occurrence of rolling a red die and a black die must have the one following outcomes (red, black)

$$\begin{aligned}\Omega = \{\, &(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6),\\ &(2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6),\\ &(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6),\\ &(4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6),\\ &(5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6),\\ &(6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)\,\}\end{aligned}$$

- There are $6 \times 6 = 36$ outcomes; they are mutually exclusive and collectively exhaustive
- But there are many other events that we can talk about…

Some 2-dice Events

- A particular outcome

  A = { (3, 6) }

- Both dice are the same

  E = { (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6) }

- The total is 5

  F = { (1, 4), (2, 3), (3, 2), (4, 1) }

- The total is prime

  G = { (1, 1), (1, 2), (1, 4), (1, 6), (2, 1), (2, 3), (2, 5), (3, 2), (3,4), (4, 1), (4, 3), (5, 2), (5, 6), (6,1), (6, 5) }

Definition of Probability

- Let P be a function that satisfies the following:

  $P(\Omega) = 1$

  all possible outcomes are accounted for

  $\forall A \subseteq \Omega : P(A) \geq 0$

  probabilities are non-negative real numbers

  $\forall \{ A, B \} \subseteq \Omega, A \cap B = \emptyset : P(A \cup B) = P(A) + P(B)$

  for any pair of events that are mutually exclusive, the union of their occurrence is the sum of their probabilities

∅ denotes the empty set, { }

It follows that:

- If P is a valid probability function for sample space Ω

  $\forall A \subseteq \Omega : P(A) \leq 1$

  Probabilities are real numbers in the range [0, 1]
- This must be true because probabilities cannot be negative (by definition), and they must sum to 1
- For every trial, an event either occurs, or does not occur

  $\forall A \subseteq \Omega : P(A^C) = 1 - P(A)$

A probability space is sometimes called 'proper' to emphasize that all of its mass is exactly accounted for, meaning that all *possible* probabilities sum to 1.0

Every events partitions Ω

- For every trial, an event either occurs, or does not occur

  $\forall A \subseteq \Omega : P(A^C) = 1 - P(A)$

- Each event $A \subseteq \Omega$ can be thought of as partitioning the probability space

Every event defines its own new, proper probability space containing two outcomes A and A^C

Mutually Exclusive Outcomes

- It is impossible for two mutually exclusive events to co-occur on the same trial
- For the 2 dice example, each of the 36 basic outcomes are mutually exclusive with each other, and the entire set is collectively exhaustive
- Therefore, one way of defining *P* is to assume that these outcomes are all equally likely:

$$EE = \{\ (1, 6)\ \}$$

$$PP(EE) = \frac{1}{|\Omega|} = \frac{1}{36} = .0278$$

This is the objective when manufacturing a fair gaming die

Compositional Events

- Events which are not in the set of mutually-exclusive, collectively-exhaustive events can be composed from them
- Compositional events are handy for grouping together certain types of events that we might be interested in
- If the function *P* describes a valid probability space, then the definition of well-formed *P* allows us to calculate *P* for mutually exclusive compositional events

$$\forall \{ A, B \} \subseteq \Omega, A \cap B = \emptyset : P(A \cup B) = P(A) + P(B)$$

- Every trial has an outcome, which may satisfy multiple events; this can be illustrated with Venn Diagrams

2 Dice Events

- Both dice are the same

  $E = \{ (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6) \}$

  $P(E) = .0278 \times 6 = .1667$

- The total is 5

  $F = \{ (1, 4), (2, 3), (3, 2), (4, 1) \}$

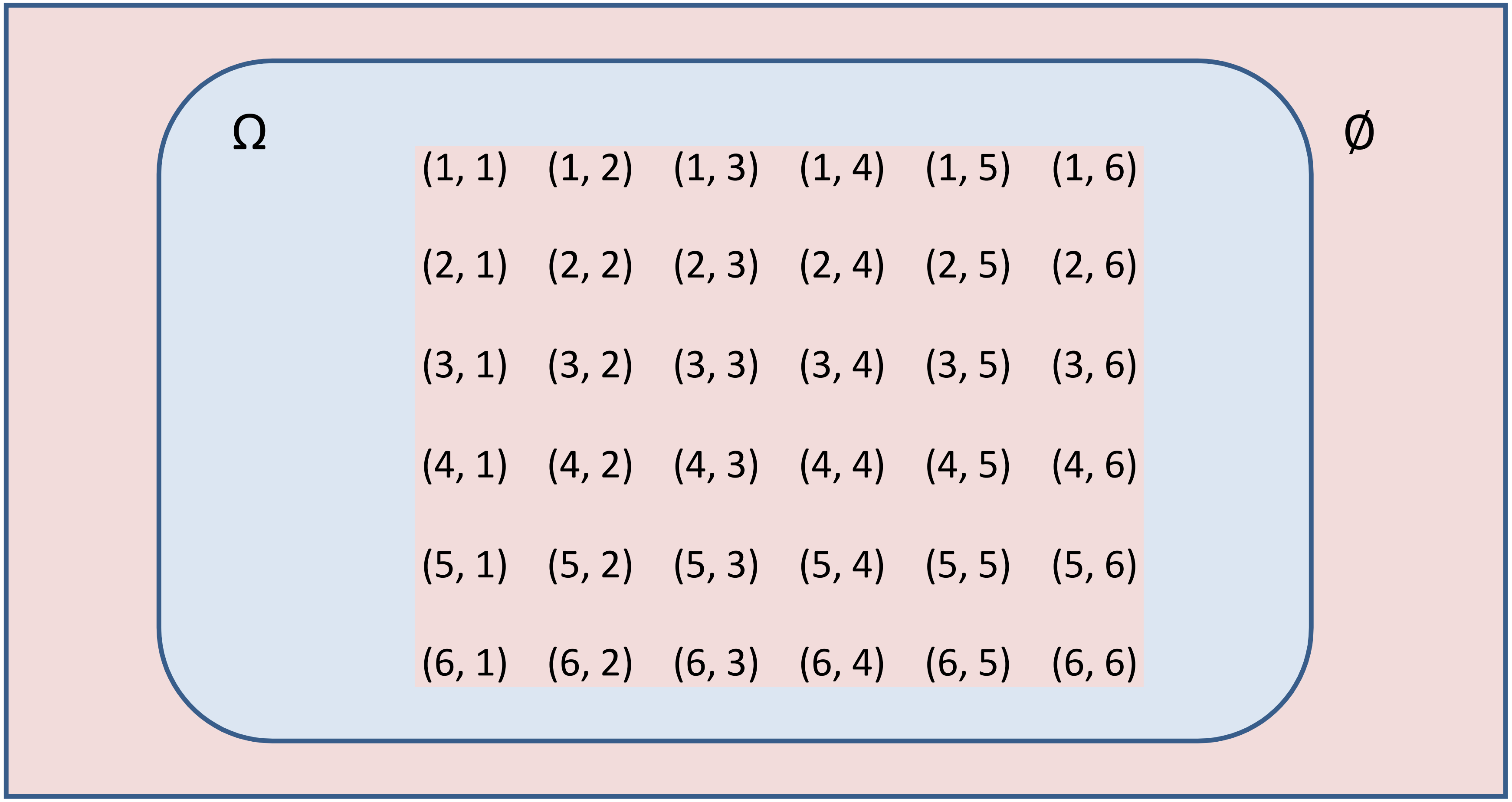
  $P(F) = .0278 \times 4 = .1111$

- The total is prime

  $G = \{ (1, 1), (1, 2), (1, 4), (1, 6), (2, 1), (2, 3), (2, 5), (3, 2), (3, 4), (4, 1), (4, 3), (5, 2), (5, 6), (6,1), (6, 5) \}$

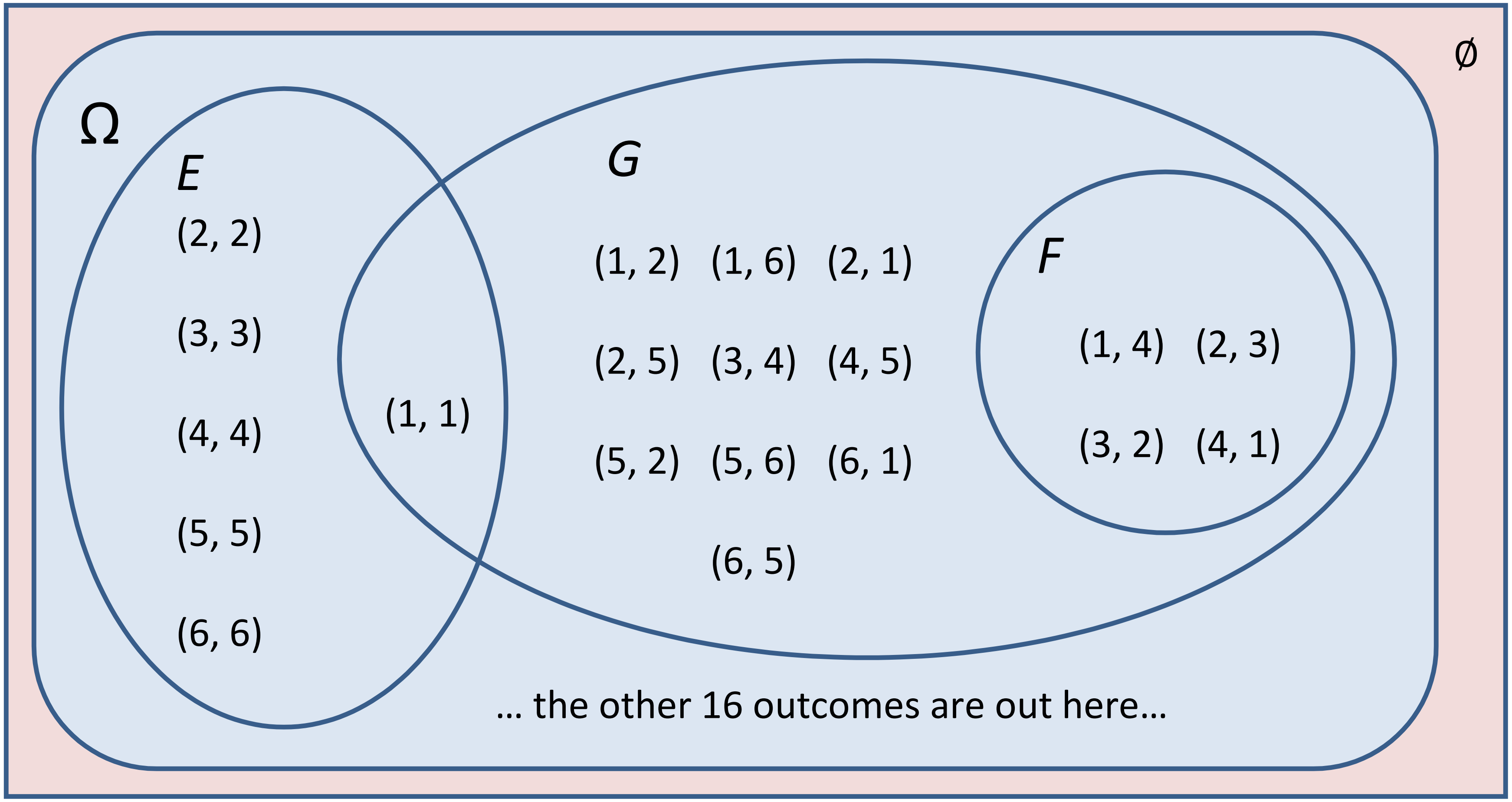
  $P(G) = .0278 \times 15 = .4167$

Outcomes in Probability Space



(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)
(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)

Event Composition



Intersecting Events

- The previous slide shows that compositional events can be mutually exclusive

  E and *F* are mutually exclusive

  $E \cap F = \emptyset$

  E and *G* are not mutually exclusive

  $E \cap G = \{ (1, 1) \}$

  F and *G* are not mutually exclusive

  $F \cap G = \{ (1, 4), (2, 3), (3, 2), (4, 1) \} = F$

More on Adding Probabilities

- We have seen how to calculate probability of P(*A* or *B*) when *A* and *B* are mutually exclusive

$$P(A \cup B) = P(A) + P(B),\ A \cap B = \emptyset$$

- If they are not, we can subtract the probability of the intersecting area

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Probability of both dice being the same, *or* their total being prime in a single trial

$$P(E \cup G) = P(E) + P(G) - P(E \cap G)$$

$$= .1667 + .4167 - .0278$$

$$= .5556$$

We got this value for

$P(E \cap G)$ by $\dfrac{|E \cap G|}{|\Omega|}$

but more discussion follows

Joint Probability

- On the previous slide we knew that $P(E \cap G) = .0278$ by noting that only 1 of the 36 mutually exclusive, collectively exhaustive outcomes is in the set intersection $E \cap G$
- More generally though, how can we compute $P(E \cap G)$ from $P(E)$ and $P(G)$?
- $P(E \cap G)$, or $P(E$ and $G)$, or $P(EG)$ is the probability that two events both occur in the same trial
- This is called the joint probability
- For mutually exclusive events, the joint probability is obviously zero:

$$\forall \{ A, B \} \subseteq \Omega, A \cap B = \emptyset : P(A \cap B) = 0$$

Joint Probability

Recall our example

$E = \{ (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6) \}$

"both dice are the same"

$F = \{ (1, 4), (2, 3), (3, 2), (4, 1) \}$

"the total is 5"

$$\forall \{ A, B \} \subseteq \Omega, A \cap B = \emptyset : P(A \cap B) = 0$$

$$E \cap F = \emptyset, \; \therefore P(E \cap F) = 0$$

The probability is zero, meaning it is not possible for both dice to be the same and for the total to be 5 on the same trial.

Joint Probability

- Perhaps it is the case that

  $$P(E \cap G) = P(E)\,P(G)$$

  Let’s try it

  $$.0278 \overset{?}{=} .1667 \times .4167$$

  $$.0278 \overset{?}{=} .0694$$

  No. This means that events *E* and *G* are not independent

Independent Events

- Independence is not the same as mutual exclusivity
  - 2 events are mutually exclusive if they cannot both occur as the outcome of a single trial
  - 2 events are independent if the occurrence of one does not affect the probability of the other occurring in the trial
- Does event A provide any information that would bias the outcome of event B?
  - If so, A and B are *not* independent events; they are dependent
- Events *E*, *F* and *G* in the 2-dice example are *not* independent of each other ({*E*, *F*}, {*F*, *G*} and {*E*, *G*})
  - Even though *E* and *F* are mutually exclusive

Adding an independent event to our example

Let's start with event *F* and try to think of an event that would be independent of *F*

$F = \{ (1, 4), (2, 3), (3, 2), (4, 1) \}$

"the total is 5"

$P(F) = .1111$

It's not so easy to come up with an event that, in the same trial, will give us no information about *F*. Such an event must meet the following criteria:

•Since F does not partition Ω equally, a event that is independent of F must partition Ω equally, so as not to bias for or against F.

•For the same reason, the event must also partition F equally.

Any ideas?

An event that is independent of *F*

“the red die shows an odd number”

H = { (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6),
(3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6),
(5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6) }

$P(H) = .5$

F = { (1, 4), (2, 3), (3, 2), (4, 1) }

“the total is 5”

$P(F) = .1111$

$H \cap F$ = { (1, 4), (3, 2) }

$\mathrm{P}(H \cap F) = .0555 \overset{?}{=} P(H)\,P(F) \overset{?}{=} .5 \times .111$ ✓

Independent Events

When two events are independent, the probability of both occurring in the same trial is

$$P(A \cap B) = P(A)\,P(B)$$

- Actually, the reverse of this is the *definition* of independence
- This is how we can test for independence of events
  - we can compare the probability $P(A \cap B)$—obtained from counting—to the product of $P(A)$ and $P(B)$. If they are equal, the events are independent

Conditional Probability

- But what if two events are not independent? How do we compute $P(A \cap B)$ from $P(A)$ and $P(B)$?
- We must know how the events are related
- $P(A|B)$ is notation for the probability of event *A*, assuming that event *B* has co-occurred in the same trial
- This is called conditional probability
- “the probability of A, given B”
- Think of a constrained probability space which contains only those outcomes which satisfy event *B*
  - *or a ‘pre-filter’ which selects only outcomes which satisfy B*

Conditional Probability

- Because the reduced sample space is limited to events which satisfy *B*, we exclude from *A* any outcomes that do not satisfy *B*: $P(A \cap B)$
- This lets us express the conditional probability in terms of the reduced sample space

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A \cap B) = P(A|B)P(B)$$

- If $P(B)$ is 0, then $P(A|B)$ is undefined

Marginal Probability

- Conditional probability introduces the idea that you might have information about *part* of a trial
- In the following equation, we are assuming that we can estimate or provide *P*(*B*) for an incomplete trial

$$PP(AA \cap BB) = PP(AA|BB)PP(BB)$$

- P(B) here is called the marginal probability

$$PP(AA \cap BB) = PP(AA|BB)PP(BB)$$

joint probability = conditional probability × marginal probability

Conditional probability and independence

- Note that conditional probability degrades gracefully in the case of independent events
- Assuming *A* and *B* are independent events:

$$PP(AABB) = PP(AA)\,PP(BB)$$
$$PP(AABB) = PP(AA|BB)\,PP(BB)$$
$$PP(AA)\,PP(BB) = PP(AA|BB)\,PP(BB)$$
$$PP(AA) = PP(AA|BB)$$

$$PP(AABB) = PP(AA)\,PP(BB)$$
$$PP(AABB) = PP(BB|AA)\,PP(AA)$$
$$PP(AA)\,PP(BB) = PP(BB|AA)\,PP(AA)$$
$$PP(BB) = PP(BB|AA)$$

If *A* and *B* are independent, then what you may know about one doesn’t affect the probability of the other

Summary of Event Probability

- $P(A^C) = 1 - P(A)$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- If $P(A \cap B) = P(A)\, P(B)$, then A and B are called independent events
- Otherwise

$$PP(AA \cap BB) = PP(AA|BB)PP(BB)$$

- Conditional probability

$$PP(AA|BB) = \frac{PP(AA \cap BB)}{PP(BB)}$$

Next Week

- Tuesday
  - Random Variables, the Chain Rule, and Probability Distributions
- Thursday
  - Project 1 due at 11:45 p.m.