Bahria University
Department of Computer Science

Adv. Natural Language Processing

Lecture 2 & 3

Instructor: Dr. Muhammad Asfand-e-yar

Previous Lecture

- What is NLP?
- Examples of NLP
- What is Special about Languages?
- Why it is hard for Computer to understand Human Language?
- Where NLP lies?
- Applications of NLP

Today's Lecture

- Regular Expressions
- Text Normalization
- Edit Distance

Basic Text Processing

Regular Expressions

ELIZA Application

User:	You are like my father in some ways.
ELIZA:	**WHAT RESEMBLANCE DO YOU SEE**
User:	You are not very aggressive, but I think you don't want me to notice that.
ELIZA:	**WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE**
User:	You don't argue with me.
ELIZA:	**WHY DO YOU THINK I DON'T ARGUE WITH YOU**
User:	You are afraid of me.
ELIZA:	**DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU**

The dialogue above is from **ELIZA**, an early natural language processing system that could carry on a limited conversation with a user by imitating the responses of a Rogerian psychotherapist

http://psych.fullerton.edu/mbirnbaum/psych101/Eliza.htm

ELIZA Application

ELIZA is a surprisingly simple program that uses pattern matching to recognize phrases like

"You are X"

and translate them into suitable outputs like

"What makes you think I am X?".

This simple technique succeeds in this domain because **ELIZA** doesn't need to know anything to mimic a Rogerian psychotherapist.

Eliza's mimicry of human conversation was remarkably successful: many people who interacted with ELIZA came to believe that it really understood them and their problems, many continued to believe in ELIZA's abilities even after the program's operation was explained. Even today such **chatbots** are a fun diversion.

Regular Expression

Important tool for describing text patterns: **the regular expression (RE)**

RE is used to specify strings that might be extracted from a document,
For example; “You are X” in Eliza above, to defining strings like $199 or $24.99 for extracting tables of prices from a document.

RE is used to turn to a set of tasks collectively called **text normalization**, in which plays an important part.

Normalizing text means converting text to a more convenient, standard form.
For example, most of what we are going to do with language relies on first separating out or **tokenizing words** from running text, the task of tokenization.

Regular Expression

Regular Expression (RE) is a standardized text search strings in computer science language for specifying text.

RE is used in every computer language, word processor, and text processing tools like the Unix tools grep or Emacs.

RE is an **algebraic notation** for characterizing a set of strings.

RE is particularly useful for searching in texts, when we have a **pattern** to search for and/or a **corpus of texts** to search through.

Regular Expression

The simplest kind of RE is a sequence of simple characters.

To search for ***woodchuck***, we type **/woodchuck/**.

OR the expression **/Buttercup/** matches any string containing the substring **Buttercup**; grep with that expression would return the line “I’m called little Buttercup”.

The search string can consist of a single character (like /!/) or a sequence of characters (like /url/).

Regular Expression

A formal language for specifying text strings

How can we search for any of these?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks

Regular Expression

Regular Expressions are Case Sensitive;

RE	Example Patterns Matched
/woodchucks/	“interesting links to woodchucks and lemurs”
/a/	“Mary Ann stopped by Mona’s”
/!/	“You’ve left the burglar behind again!” said Nori
Some Simple regular expression search	

Regular Expression

Regular Expressions are Case Sensitive;

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	“Woodchuck”
/[abc]/	‘a’, ‘b’, or ‘c’	“Hast du gut gelernt?”
/[1234567890]/	any digit	“plenty cubes having 1”
The use of the brackets [ ] to specify a disjunction of characters.		

Therefore, it can also
/[ABCDEFGHIJKLMNOPQRSTUVWXYZ]/
Means “any Capital Letter”

Regular Expression

Regular Expressions are Case Sensitive;

RE	Match	Example Patterns
/[A-Z]/	an upper case letter	“we should call it ‘Gut Gemacht”
/[a-z]/	a lower case letter	“the Bean was impatient”
/[0-9]/	a single digit	“Kapital 1: Text Schrieben”
The use of the brackets [ ] to specify a disjunction of characters.		

Regular Expression

The square braces can also be used to specify what a single character should not be used in a string by use of the caret **^**.

If the caret **^** is the first symbol after the open square brace [, the resulting pattern is negated.

For example, the pattern /[^a]/ matches any single character (including special characters) except a.

This is only true when the caret is the first symbol after the open square brace.

If it occurs anywhere else, it usually stands for a caret symbol

Regular Expression

Regular Expressions with condition “Not a substring”;

RE	Match	Example Patterns
/[^A-Z]/	not an upper-case letter	“Am Himmel”
/[^Ss]/	neither ‘S’ nor ‘s’	“Hast du gut gelernt?”
/[^\.]/	not a period or dot	“our resident”
/[e^]/	either ‘e’ or ‘^’	“look up ^ now”
/a^b/	the patter ‘a^b’	“look up a^b now”
Uses of caret/cap “^”		

Regular Expression

How can we talk about optional elements, like an optional “s” in “woodchuck” and “woodchucks”?

We can’t use the square brackets, because while the “[ ]” allow us to say, “s or S”, they don’t allow us to say, “s or nothing”.

Therefore, we use the question mark /?/, which means “the preceding character or nothing”.

The question mark means that “zero or one instances of the previous character”.

Regular Expression

Regular Expressions with Optional Conditions";

RE	Match	Example Patterns
/woodchucks?/	woodchuck or woodchucks	"woodchuck"
/colou?r/	color or colour	"colour"
Uses of question mark "?"		

RE	Match	Example Patterns
/beg.n/	any character between beg and n	"begin" , "beg'n" , "begun"
Uses of the period or dot "." to specify any character		

Regular Expression

How can I distinguish between cat and dog?

Since we can't use the square brackets to search for "cat or dog"
Why can't we say **/[catdog]/**?

We need a new operator, the disjunction operator, also called the pipe symbol **|**.

The pattern **/cat|dog/** matches

- either the string **cat**
- or the string **dog**

Regular Expression

Disjunction

Woodchucks is another name for groundhog!

- The pipe | for disjunction

RE	Example Pattern
/groundhog|woodchuck/	“groundhog” or “woodchuck”
/yours|mine/	“yours" or “mine”
/a|b|c/	= [abc]
/[gG]roundhog | [Ww]oodchuck/	

Regular Expression

Sometimes we need to use the disjunction operator in the midst of a larger sequence.

For example, suppose I want to search for information about pet fish for my cousin David. **How can I specify both guppy and guppies?**

Is it possible to express the above as **/guppy|ies/**?

No, because that would match only the strings **guppy** and **ies**.

Regular Expression

Precedence:

This is because sequences like guppy take precedence over the disjunction operator |.

To make the disjunction operator apply only to a specific pattern, then use the parenthesis operators (and).

Therefore, the pattern **/gupp(y|ies)/** would specify that we meant the disjunction only to apply to the suffixes **y** and **ies**.

Regular Expression

Operator Precedence

This idea that one operator may take precedence over another, use parentheses to specify what we mean, is formalized by the operator precedence hierarchy for regular expressions.

The following table gives the order of RE operator precedence, from highest precedence to lowest precedence.

1	Parenthesis	()
2	Counters	* + ? {}
3	Sequences and anchors	the ^my end$
4	Disjunction	|

Thus, because Counters have a higher precedence than Sequences

Regular Expression

For example; match repeated instances of a string.

Column 1 Column 2 Column 3

/Column [0-9]+ */

Will not match any number of columns; instead, it will match a single column followed by any number of spaces.

The star here applies only to the space that precedes it, not to the whole sequence.

With the parentheses, the expression is;

/(Column [0-9]+ *)*/

to match the word Column, followed by a number and optional spaces, the whole pattern repeated any number of times.

Regular Expression

Construct a Finite State Machine for a language of certain sheep?

The string that look like the following:

baa

baaa

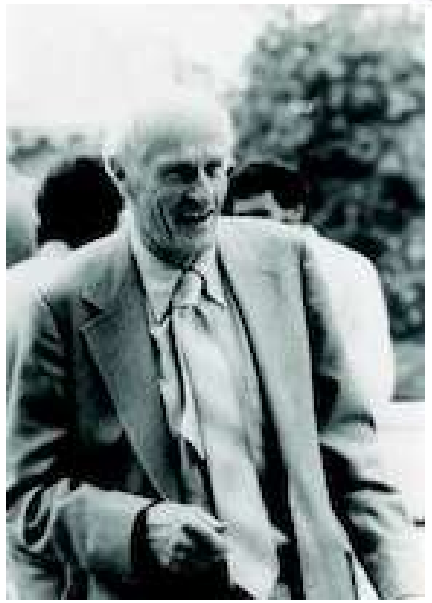
baaaa

baaaaa

…

Regular Expression

The string that look like the following:
baa
baaa
baaaa
baaaaa
…


Stephen C Kleene

On the above string we will apply Kleene Closure property;
baa* or ba^{+}

Regular Expression

Kleene Closure property.

baa* or ba^+

Therefore, it can be expressed as /baa*/ or /[baa]*/

1. /baa*/ => means one ***b***, one ***a*** followed by NULL or more ***a***s
2. /ba+/ => means one ***b*** *followed by* one ***a*** or more ***a***s
3. /[baa]*/ => means zero or more ***b***s or ***a***s (incorrect according to the given language)

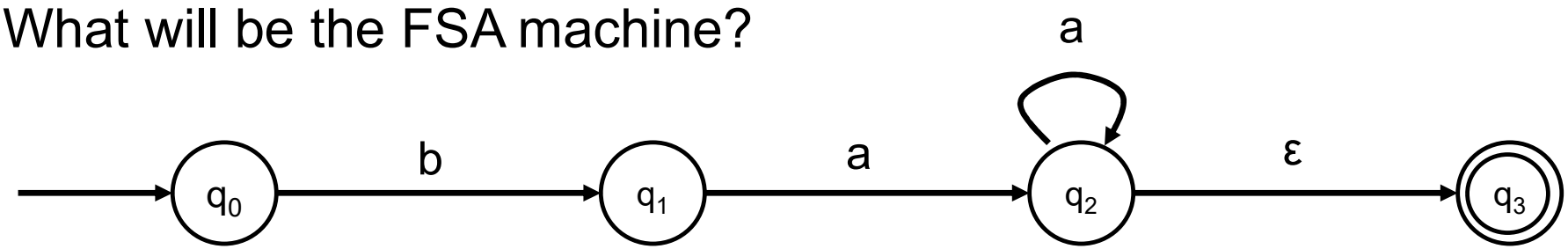
Therefore, according to language the correct expression will be /baa*/

Regular Expression

Kleene Closure property.

baa* or ba^{+}

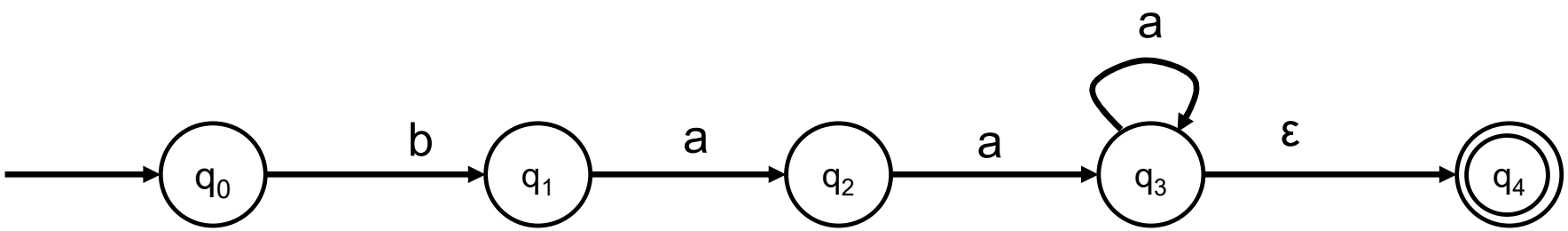
What will be the FSA machine?



Regular Expression

Kleene Closure property.

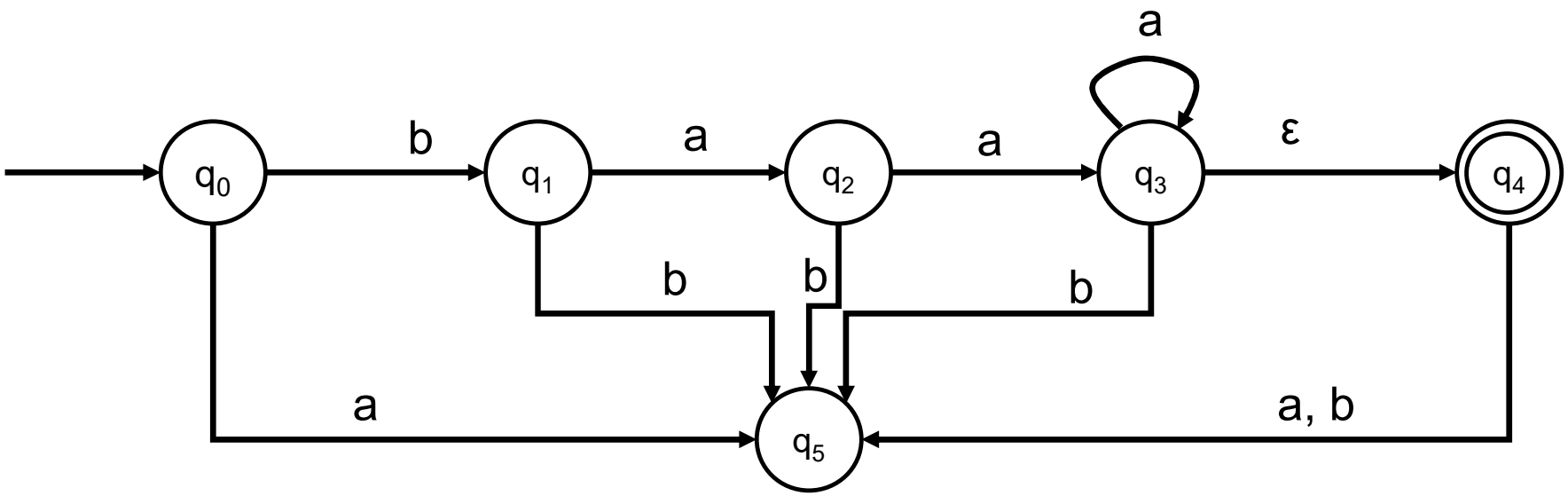
baa* or ba^{+}



Regular Expression

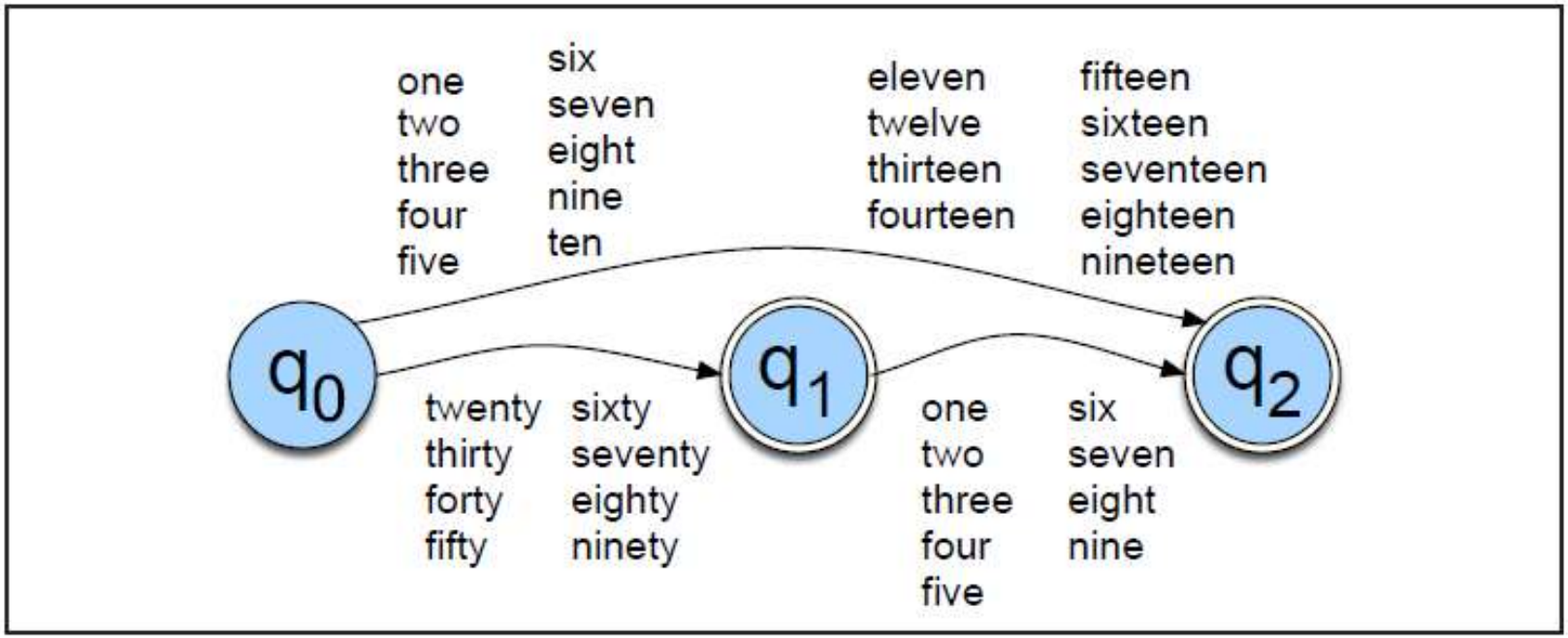
Kleene Closure property.

baaa* or baa^{+} $L(M) = \{baa, baaa, baaaa, baaaaa, baaaaaa, \ldots\}$



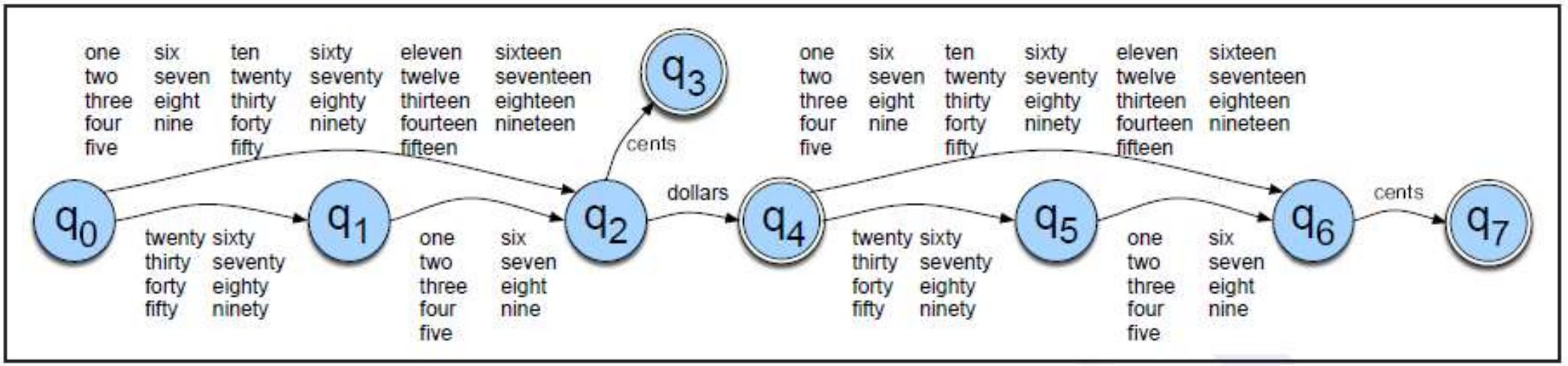
Regular Expressions

An FSA for the words for English numbers 1–99.



Regular Expressions

An FSA for the words for English numbers 1–99 in Dollars and/or Cents.



Regular Expressions:

RE=> ? , * , +

Kleene *, Kleene +

RE	Matches	Example Pattern
/colou?r/	Optional previous char	color , colour
/oo*h!/	0 or more of previous char	oh! , ooh! , oooh! , ooooh!
/o+h!/	1 or more of previous char	oh! , ooh! , oooh! , ooooh!
/baa*/	b and 1 or more a	baa , baaa , baaaa , baaaaa
/ba+/	b and 1 or more a	baa , baaa , baaaa , baaaaa

Regular Expressions

RE - Anchors => ^ , $

RE	Matches	Example Pattern
/^[A-Z]/	in the beginning capital letter	“Palo Alto”
/^[^A-Za-z]/	no alphabet in the beginning	“1” or “ ‘Hello’ ”
/\.$/	period “.” in the end	“The end.”
/.$/	any thing in the end	“The end?” or “The end!”

Example “the”

Find me all instances of the word “the” in a text.

/the/

Misses capitalized examples

/[tT]he/

won’t treat underscores and numbers as word (the_ or the25)

/\b[tT]he\b/

Incorrectly returns other or theology

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

It is not being mentioned that “the” word should be in beginning of a line.

Example “the”

There are also two other anchors:

- \b matches a word boundary
- \B matches a non-boundary

For example, /\b99\b/ will match the string 99 in

“There are 99 bottles of beer on the wall”

(because 99 follows a space)

But not 99 in

“There are 299 bottles of beer on the wall”

(since 99 follows a number).

But it will match 99 in $99

(since 99 follows a dollar sign ($), which is not a digit, underscore, or letter).

Example “the”

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

“the” word should be in beginning of a line or in end of a line

/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/

Example “the”

The process we just went through was based on fixing two kinds of errors

Matching strings that we should not have matched (there, then, other)

False positives (Type I)

Not matching things that we should have matched (The, the)

False negatives (Type II)

Example “the”

In NLP we are always dealing with these kinds of errors.

Reducing the error rate for an application often involves two antagonistic efforts:

- Increasing accuracy or precision (minimizing false positives)
- Increasing coverage or recall (minimizing false negatives).

Complex Example

Let's try out a more significant example of the RE.

For example; build an application to help a user buy a computer on the Web.

- The user might want

"any machine with more than 6 GHz and 500 GB of disk space for less than $1000".

- To do this kind of retrieval, initially analyze expressions like 6 GHz or 500 GB or Mac or $999.99.

In the rest of the section some simple regular expressions will be analyzed for this task.

Complex Example

First, let's complete RE for prices.

RE for a dollar sign followed by a string of digits:
/$[0-9]+/

$ is for the end

Decimal point and two digits afterwards
/$[0-9]+\.[0-9][0-9]/

This pattern only allows $199.99 but not $199

Complex Example

Make the cents optional and to make sure a word boundary:

/\b$[0-9]+(\.[0-9][0-9])?\b/

How about specifications for processor speed?

Here's a pattern for that:

/\b[0-9]+␣*(GHz|[Gg]igahertz)\b/

Allowing optional fractions again *"5.5 GB"*;

/\b[0-9]+(\.[0-9]+)? *(GB|[Gg]igabytes)?\b/

␣ means White Spaces

Regular Expression

RE /{3}/ means
“exactly 3 occurrences of the previous character or expression”.

Therefore,
/a\.{24}z/
will match ***a*** followed by ***<u>24 dots</u>*** followed by ***<u>z</u>***

/a\.{24, 30}z/
will match ***a*** followed by ***<u>24 dots OR upto 30 dots</u>*** followed by ***<u>z</u>***

/a\.{24, }z/
will match ***a*** followed by **<u>at least *24 dots*</u>** followed by ***<u>z</u>***