Ling/CSE 472:
Introduction to Computational Linguistics

3/30/2023
Regular Expressions

Overview

- Regular expressions
- Words, lemmatization, tokenization
- Reading questions

Regular Expressions

- Textbook: Presented as a tool for searching (& replacing)
- Here:
  - Regular languages as a class of languages
  - Regular expressions as a way to describe regular languages (with asides about FSAs)
  - Search

Formal languages

- From the point of view of formal languages theory, a language is a set of strings defined over some alphabet.
- Chomsky hierarchy: a description of classes of formal languages:
  - regular < context-free < context-sensitive < all languages
- Languages from a single level in the hierarchy can be described in terms of the same formal devices.

Three views on the same object

Regular languages can be described by regular expressions and by finite-state automata.

- Regular language: a set of strings
- Regular expression: an expression from a certain formal language which describes a regular language
- Finite-state automaton: a simple computing machine which accepts or generates a regular language

Formal definition of regular languages: symbols

- ϵ is the empty string
- ø is the empty set
- Σ is an alphabet (set of symbols)
- Examples of alphabets:
  - Σ = {a,b,!}
  - Σ = {c,a,t,d,o,g,f,x}
  - Σ = {cat, dog, fox}
  - Σ = {1, 3, 5, 7}

Formal definition of regular languages

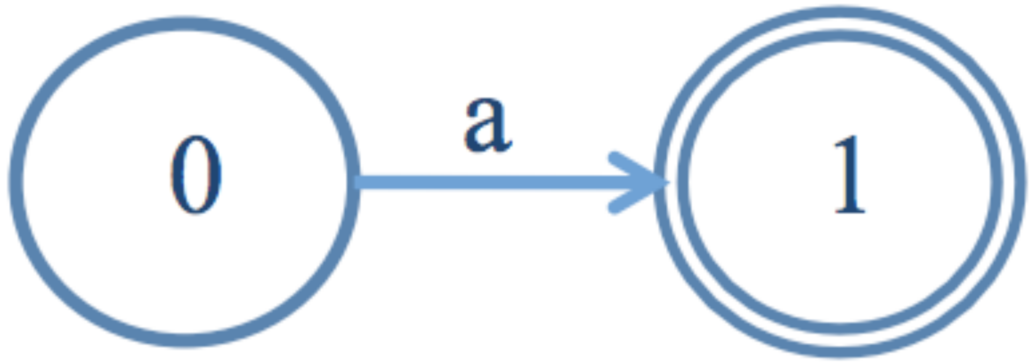
The class of regular languages over Σ is formally defined as:

- ø is a regular language
- ∀a ∈ Σ ∪ ϵ, {a} is a regular language
- If L1 and L2 are regular languages, then so are:
    - L1 · L2 = {xy | x ∈ L1, y ∈ L2} (concatenation)
    - L1 ∪ L2 (union or disjunction)
    - L1* (Kleene closure)

(Jurafsky & Martin 2009:39)

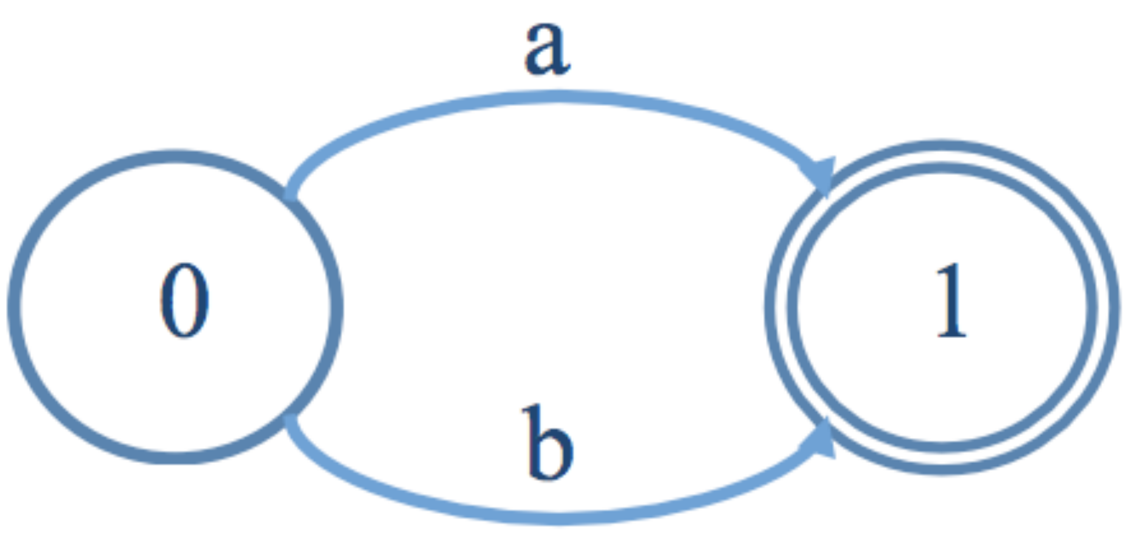
Examples: Single Character

- Alphabet: Σ = {a}
- Language (set of strings): L = {a}
- Regular expression: /a/
- Finite state machine:



Examples: Disjunction/union

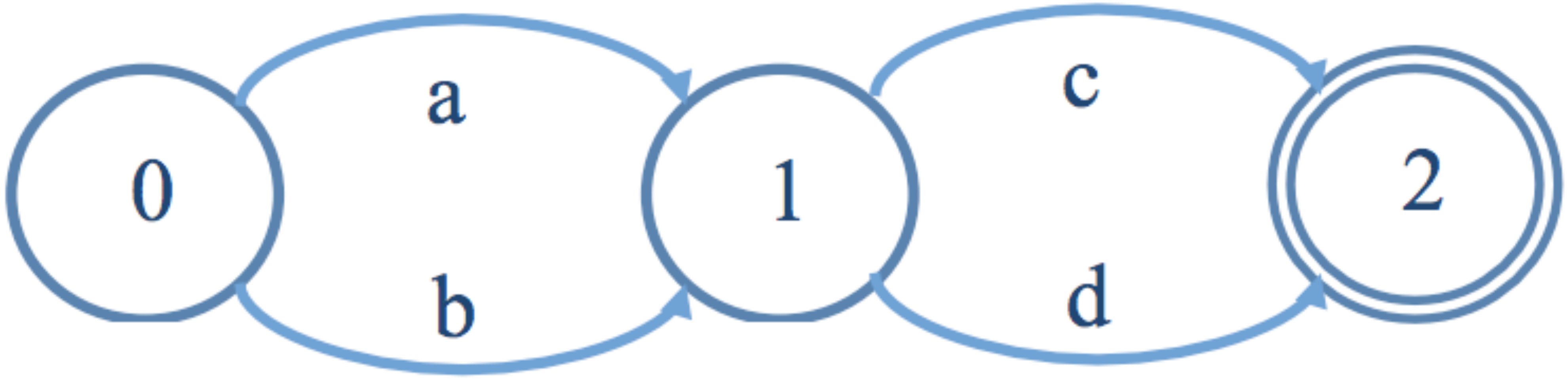
- Alphabet: Σ = {a, b}
- Language (set of strings): {a, b}
- Regular expression: /(a|b)/
- Finite state machine:



- This language/regexp/FSA also reflects the union of two languages: {a} ∪ {b}

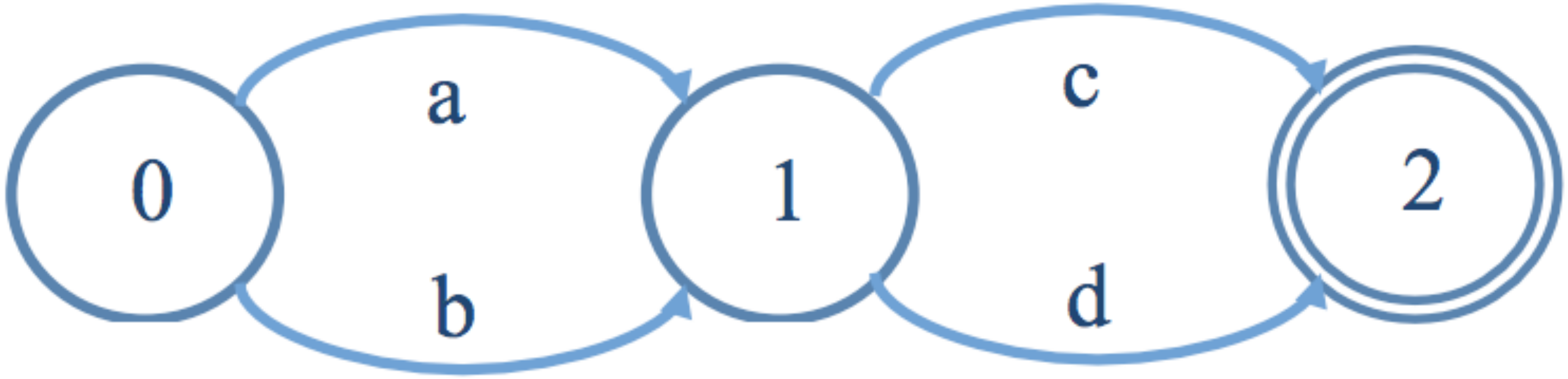
Examples: Concatenation

- Σ = {a, b, c, d}
    - L = {ac, ad, bc, bd}
    - Regular expression: /(a|b)(c|d)/
    - Finite state machine:



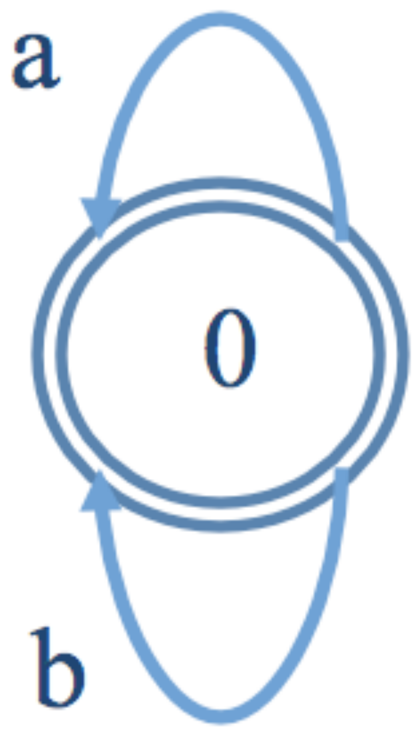
Examples: Concatenation

- Note: This language/regexp/FSA also reflects the concatenation of two languages:
    - L1 = {a, b} · L2 = {c, d}



Examples: Kleene closure

- Σ = {a, b} (again)
- L = {ϵ, a, b, aa, bb, ab, ba, bb, ...}
- Regular expression: /(a|b)*/
- Finite state machine:



Regular expression examples

- abc
- a|bc
- (a|bb)c
- a[bc]
- a*b
- a?b

Search with regular expressions

- For each input (usually a ‘line’), does the regular expression match any part of the input?
- In other words, does the line contain a string of the regular language?
- Two more special symbols:
    - ^ : beginning of line
    - $: end of line

Regular expression examples

- abc
- a|bc
- (a|bb)c
- a[bc]
- a*b
- a?b
- [^a]*th[aeiou]+[a-z]*
- ^a
- ^a.*z$
- \bthe\b

Regular expressions: An extension

- Parentheses allow you to ‘save’ part of a string and access it again.
- In general:
    - ( ) creates a group that can be referenced
    - \n refers back to the *n*th group in a regular expression
- For example, to search for repeated words: /([a-z]+)\1 \1/
- That will find any group of one or more lower case alphabetic characters repeated twice (and separated by a space): ‘I called kitty kitty kitty.’

Overview

- Regular expressions
- Words, lemmatization, tokenization
- Reading questions

Words

- What’s a word?
  - Linguistically
  - Practically (for NLP)
- How do you count words
  - In a corpus
  - In a “language”

Examples of words_with_spaces

Total Results: 0

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Examples of tokens that are more than one word

Total Results: 0

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Tokenization

- Taking larger segments of text and breaking it into smaller ones
  - Documents -> sentences
  - Sentences -> words
- What makes each of those hard in edited, written English?
- What challenges do you see in other languages?
- What challenges do you see in other genres?

Lemmatization

- Lemma: ‘a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense.’ (p.11)
- How does this relate to stem? Lexical entry?
- What might this be useful for?
- Lemmatization as a task:
    - What’s the input?
    - What’s the output?
    - How would you evaluate a lemmatizer?

Overview

- Regular expressions
- Words, lemmatization, tokenization
- Reading questions

Reading questions

- A lot of the text examples for matches found with regex give spaces as underscores, like Party_of_5. Do these texts actually write underscores for spaces? Or is it just that the computer program interprets them as such?

Reading questions

- What does ‘zero-width’ mean?
  - /^(?!Volcano)[A-Za-z]+/

Reading questions

- For regex like /$[0-9]+/, they say that a parser is usually 'smart' enough to understand that this use of $ does not indicate and end-of-line. yI sort of makes sense, as it's at the beginning followed by more characters of a single string, but I'm not sure why it makes as much sense to a parser to see that as well. How would it derive that from the context?

Reading questions

- I am still having some trouble with the concept of precedence. I understand how it determines the order in which the operators will apply, but I am still wondering what is it about each operator that causes them to be ordered in this way? Why is it that sequences have a higher precedence than disjunction, for example?

Reading questions

- How does the underlying implementation of regex work and how may they impact decisions we make when creating/using regex. For example, are there certain implementation details that make certain operators less performant than others?

Reading questions

- In the process of text normalization, we need to find the stem of the word through stemming. How does algorithm determine the stem of a word as there are many different prefix and postfix that can change the type of words without changing its meaning. Do we need to manually code them or we can a more general algorithm to process that.

- The reading mentions that a simpler and cruder alternative to lemmatization algorithms, called stemming, can sometimes be useful alternative, but tend to commit a lot of errors. In what kind of situation would this be preferable to a more advanced algorithm? How do complex lemmatization algorithms prevent the same kinds of errors from occurring?

Reading questions

- The reading mentioned that during tokenization we want to keep punctuation in some situations such as "cap'n" but not with "we're" but I don't see how a computer would differentiate between these different situations. Do contractions like "we're" need to be hard-coded in order for the computer to know to tokenize it in a certain way, or is there some more complex algorithm the computer can use based on pre-coded knowledge of the language of the text being normalized?

- I can't really think of a way to do this besides hard-coding without training some language model based on human input data.

Reading questions

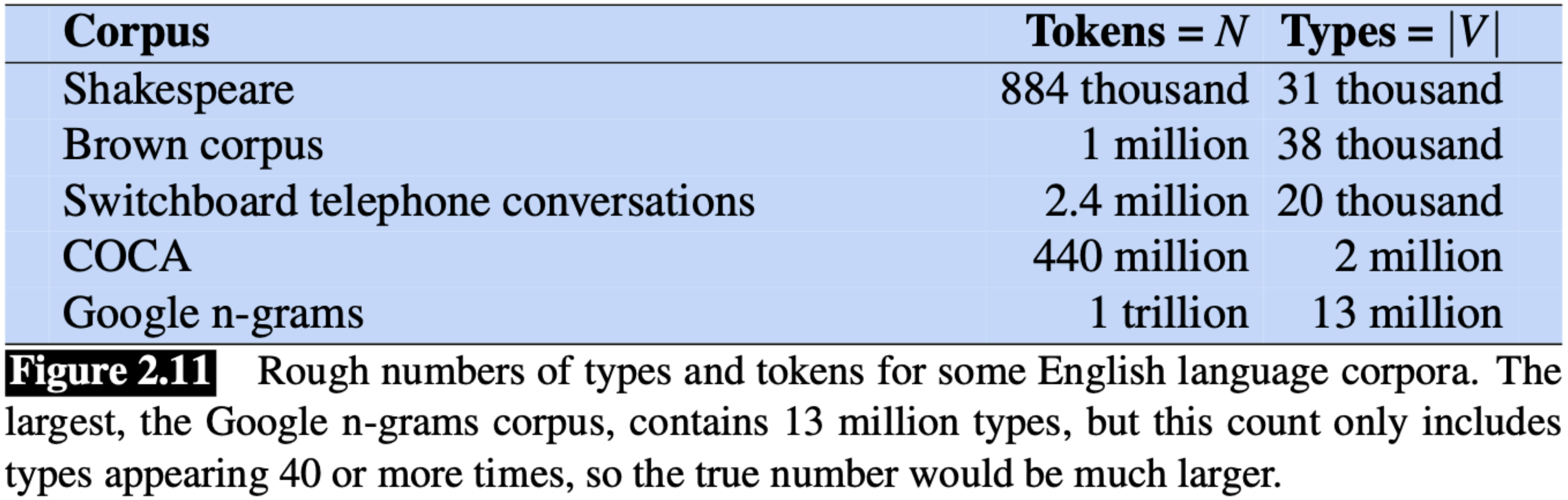
- The most confusing part of the reading was probably the section about BPE; what's the purpose of having sequences of letters that aren't morphemes (such as "ne") in the vocabulary? And what are some examples of when being unaware of the motivation/situation/collection process, etc. of language data would be an obstacle?

Reading questions

- I am interested where the data in corpora commonly comes from, how and how often/recently it is collected, and if there are filters used when collecting data for some corpora which sort out certain texts.

Reading questions

- I found Herdan's Law/Heaps' Law quite hard to grasp. I know it describes the relationship between the number of word types and the number of word tokens but what's the significance of it? Also, does it apply to all languages or just English?



Corpus	Tokens = N	Types = $\|V\|$
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google n-grams	1 trillion	13 million

Figure 2.11 Rough numbers of types and tokens for some English language corpora. The largest, the Google n-grams corpus, contains 13 million types, but this count only includes types appearing 40 or more times, so the true number would be much larger.

(From J&M 2023, Ch 2)

Look-ahead

- Assignment 1

Computational linguistics/NLP in the news

- https://futureoflife.org/open-letter/pause-giant-ai-experiments/
- Response: https://medium.com/@emilymenonbender/policy-makers-please-dont-fall-for-the-distractions-of-aihype-e03fa80ddbf1