Figure 1: Course instructors: Prof. Qun Liu (left), Dr. Valentin Malykh (right).

1 About the authors

Professor **Qun Liu** - Chief Scientist of Speech and Language Computing, Huawei Noah's Ark Lab. Prof. Liu received his PhD degree from Peking University in 2004 and held Professor position at Institute of Computing Technology, Chinese Academy of Sciences, and at Dublin City University. Qun Liu has almost 30 years of experience in NLP area and about 500 published papers.

Doctor **Valentin Malykh** - Senior Research Scientist in Speech and Semantics Lab, Huawei Noah's Ark Lab. Valentin had written his thesis at Moscow Institute of Physics and Technology and received his PhD degree at Institute for Systems Programming, Russian Academy of Sciences in 2019. Valentin has 8 years of industry experience, including Yandex as a Machine Learning Engineer and VK.com as Applied Scientist. Dr. Malykh has published more than 30 papers.

2 Introduction

Natural Language Processing (NLP) is a domain of research whose objective is to analyze and understand human languages and develop technologies to enable human machine interactions with natural languages. NLP is an interdisciplinary field involving linguistics, computer sciences and artificial intelligence. The goal of this course is to provide students with comprehensive knowledge of NLP. Students will be equipped with the principles and theories of NLP, as well as various NLP technologies, including rule-based, statistical and neural network ones. After this course, students will be able to conduct NLP research and develop state-of-the-art NLP systems.

2.1 Research questions and NLP tasks

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the

interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

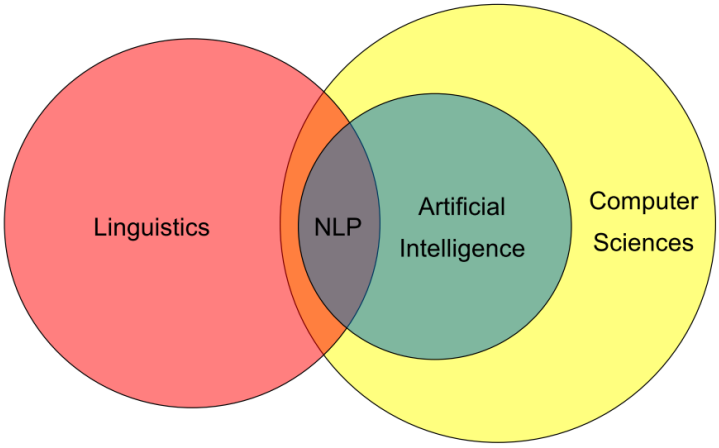

Figure 2: How NLP relates to the close entities.

The are several other names for NLP which are in use simultaneously (see Fig. 2 for their relationship). Each nom-de-plume has its own specific nuances of meaning. **Computational Linguistics** is more regarded as a branch of Linguistics, whose main purpose is to understand the mechanism of human languages by means of computing. While **Natural Language Processing** is a branch of computer sciences and artificial intelligent, whose main purpose is to develop technologies to enable human-computer interactions using human languages. **Natural Language Understanding** is also used as a synonym of NLP. It is one of the two main challenges in Natural Language Processing, while the other is **Natural Language Generation**. At last **Human Language Technologies** mainly refer to NLP technologies, but may also include other language related technologies, include speech technologies, optical character recognition (OCR), computer typesetting, etc.

2.2 Understanding of human languages

We are getting used to the fact that human beings can understand each other using language communication. Although it is a natural result of evolution for human to obtain the language competence. It seems to be a miracle due to language overwhelming complexity. No other species in this planet can use languages at the degree as humans do. For example, our far relatives apes and monkeys could learn some simplified human language, but for example monkeys and apes seem to not understand each other languages while we can learn other languages with ease or at least with some effort. The mechanism behind human languages is not fully discovered. Understanding human languages by computer

is difficult, since they seems to be more complex than typical algorithms a computer works with.

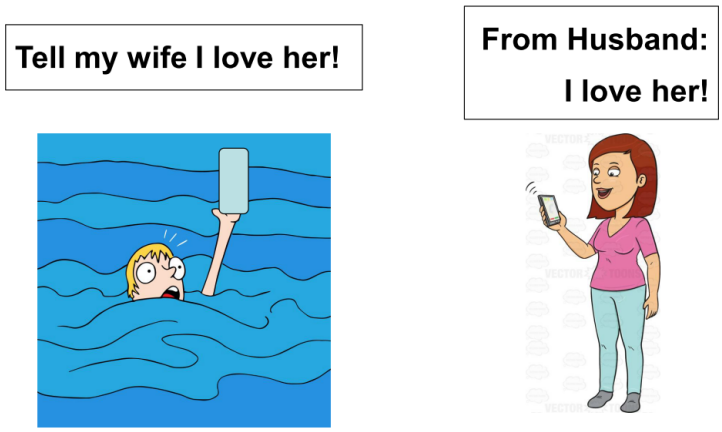

Figure 3: The context is crucial.

At Fig. 3 we could see an example of a misunderstanding. The indirect speech is misused and the meaning of a man's phrase is completely lost.

3 The ways of NLP research

Unlike linguists who develop numerous theories to explain the language mechanisms, NLP researchers try to simulate human language behaviors by computing, not necessary to understand the language mechanisms.

There were three main periods in NLP development. At the early stage of it, in **1960-1990s**, everything was based on the rules. The researches created more and more sophisticated rule sets for more and more complex problems. Relatively simple rule set stands behind famous in its day ELIZA chat-bot. It seems to be one of the first (if not the first) chat-bots, and it is pretty impressive even now. You could find some versions of it running at the Internet.

The next stage begun with more powerful computational engines **in 1990s** and it lasted **up to 2010s**. These were statistical models. The most famous models are Support Vector Machine and Decision Tree (with their variants). These models could make use of some data and generalize over it.

And the current period, which begun **in 2010s**. This is an era of Deep Learning. The deep learning models are also statistical, but since the computational power has grown significantly, especially with rise of graphical processing units, these models could handle the abundance of data. The data has become available conveniently with the triumph of the Internet on our planet.

It would be incorrect to say that the previously developed techniques are completely lost in time. They are still in use in some specific cases, for example

in the case of low resource computational devices or in case where we need to be to understand why an answer is like that (for the rules and decision trees).

Let us try to define a language. It could be defined as the set of sentences which can be accepted by the speakers of that language. It is not possible to define a natural language by enumerate all the sentences, because the number of sentences in a natural languages is infinite. There are two feasible ways to define a language with infinite sentences: by a grammar or by an automaton.

Let us first define a grammar. A grammar G is defined as a finite set of rules, and, a mechanism to generate word sequences by applying the rules in G in a finite number of time steps.

A sentence of a language is defined as: a word sequence S is called a sentence of a language L if and only if S belongs to L.

Given a grammar G, a language L could be defined by G as: a word sequence S is a sentence of L if and only if S can be generated by G.

Next, we define an automaton. An automaton A is a abstract machine which: Takes a symbol sequence S as input, and determines if A will *accept* or *reject* S; has a finite number of states and a finite number of actions; at each time step, S is in a state, and points to a position in S; the current state and current symbol determines the action which A will execute, which determines the next state of A and the next position of S where A will point to; given a input S, A will run until it stops, and the final state of A determines if A will *accept* or *reject* S.

A language L can be defined by an automaton A as: a word sequence S is a sentence of L, if and only if, when we input S to A, A will stop in a finite number of time steps at an *accept* state.

3.1 Chomsky hierarchy of formal grammars

The Chomsky hierarchy (occasionally referred to as the Chomsky–Schützenberger hierarchy) is a nested hierarchy of classes of formal grammars. This hierarchy of grammars was described by Noam Chomsky in 1956. It is also named after Marcel-Paul Schützenberger, who played a crucial role in the development of the theory of formal languages.

3.1.1 Formal grammars

- A formal grammar G is a quadruple $\{N, T, S, R\}$:
  - R: a finite set of production rules (left-hand side $\rightarrow$ right-hand side, or LHS $\rightarrow$ RHS), where each side consists of a finite sequence of the symbols from N, T or $\{S\}$
  - N: a finite set of non-terminal symbols (indicating that some production rule can yet be applied),
  - T: a finite set of terminal symbols (indicating that no production rule can be applied),
  - S: a start symbol (a distinguished non-terminal symbol).

Figure 4: Noam Chomsky

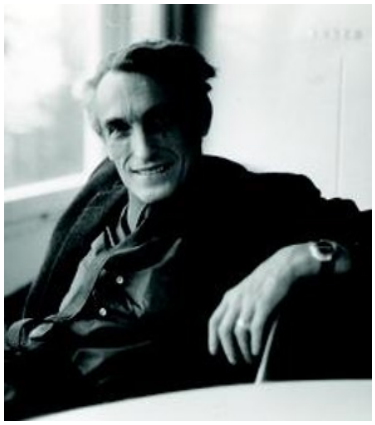

Figure 5: Marcel-Paul Schützenberger

A *formal grammar* G provides an axiom schema for a *formal language* L, which is a set of finite-length sequences of symbols that may be constructed by applying *production rules* to another sequence of symbols, which initially contains just the *start symbol*. A *production rule* may be applied by replacing an occurrence of the symbols on its left-hand side (LHS) with those that appear on its right-hand side (RHS).

A sequence of rule applications is called a *derivation*. A formal grammar G defines a formal language L: all sequences of symbols consisting solely of terminal symbols which can be reached by a derivation from the start symbol.

Let us look at the example. The language $\{a^n b^n\}$ (i.e. n copies of a followed by n copies of b) can be defined by the following grammars:

Terminals: $\{a, b\}$
Nonterminals:$\{S, A, B\}$
Rules:
$S \to AB$
$S \to \epsilon$
$S \to aS$
$S \to b$

Terminals: $\{a, b\}$
Nonterminals:$\{S\}$
Rules:
$S \to aSb$
$S \to \epsilon$

where ϵ is an empty string. These grammars are equivalent although they differ in the number of rules, the rules themselves and even the set of non-terminals.

Noam Chomsky has defined a hierarchy of grammars. It is showed in Tab. 1. Let us describe each type in more details.

3.1.2 Type 0 grammars

These grammars are also called Unrestricted Phrase Structure Grammars. A Type 0 Grammar generates a Recursively Enumerable Language. A Recursively

Grammar	Languages	Production Rules	Examples
Type 0	Recursively Enumerable	$\alpha A\beta \rightarrow \delta$	
Type 1	Context Sensitive	$\alpha A\beta \rightarrow \alpha\gamma\beta$	$L = \{ a^n b^n c^n \mid n > 0\}$
Type 2	Context Free	$A \rightarrow \alpha$	$L = \{ a^n b^n \mid n > 0\}$
Type 3	Regular	$A \rightarrow a$ or $A \rightarrow aB$	$L = \{ a^n \mid n > 0\}$

Table 1: Chomsky's hierarchy of grammars.

Enumerable Language L said to be semi-decidable by a Turing Machine T:

- For any sentence S in L, T will accept it in a finite number of time steps.
- For a sentence S not in L, it is not guaranteed that T can reject it in a finite number of time steps.

3.1.3 Type 1 grammars

A grammar of this type is called Context Sensitive Grammar. A Type 1 Grammar generates a Context Sensitive Language. A Context Sensitive Language is decidable by a Linear Bounded Automaton and the complexity of this decision problem is NP-complete. It is not practical to use Type 1 Grammars in NLP because of its time complexity.

3.1.4 Type 2 grammars

These are Context Free Grammars. A Type 2 Grammar generates a Context Free Language. A Context Free Language is decidable by a Pushdown Automaton and the complexity of this decision problem is polynomial. Type 2 Grammars are the theoretical basis of all programming languages. Type 2 Grammars are commonly used in NLP, however, natural languages are not context free languages actually.

3.1.5 Type 3 grammars

And finally there are Regular Grammars. A Type 3 Grammar generates a Regular Language. A Context Free Language is decidable by a Finite State Automaton / Machine and the complexity of this decision problem is linear. Type 3 Grammars are the theoretical basis of the lexical analyzers of all programming languages. Type 3 Grammars are very broadly used in NLP for many different purposes.

3.1.6 Regular expressions

Regular Expressions are very useful tools which are supported by most of the modern programming languages and text editors. Regular Expression is equivalent to a Regular Grammar, and vice versa.

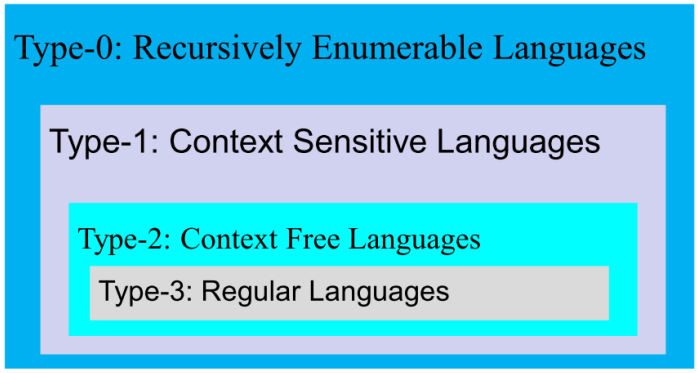

Figure 6: Set inclusions described by the Chomsky hierarchy

Grammar	**Languages**	**Automaton**	**Decidability and Complexity**
Type 0	Recursively Enumerable	Turing Machine	Semi-Decidable
	Recursive	Decider, or Total Turing Machine	Decidable
Type 1	Context Sensitive	Linear Bounded Automaton (LBA)	NP Complete
Type 2	Context Free	Pushdown Automaton (PDA)	Polynomial
	Deterministic Context Free	Deterministic Pushdown Automaton (PDA)	Linear
Type 3	Regular	Deterministic / Nondeterministic Finite State Machine (FSM)	Linear

Table 2: Grammars and automata

3.1.7 Other grammar classes

There are so called Mild Context Sensitive Grammars. These are grammar classes which define subsets of Context Sensitive Languages, but beyond Context Free Languages. Examples: Index Grammars (IGs), Tree Adjoin Grammars (TAGs).

3.2 Automata

There is a special automaton, which has a significant impression on the whole field of computation, and the natural language processing as its part. It is Turing machine. A Turing machine is presented on Fig. 7ab. Alongside it a portrait of Alan Turing in young age is presented at Fig. 7c. A Turing machine consists of:

- A *tape* divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol and some other symbols. The *tape* is assumed to be arbitrarily extendable to the left and to the right.

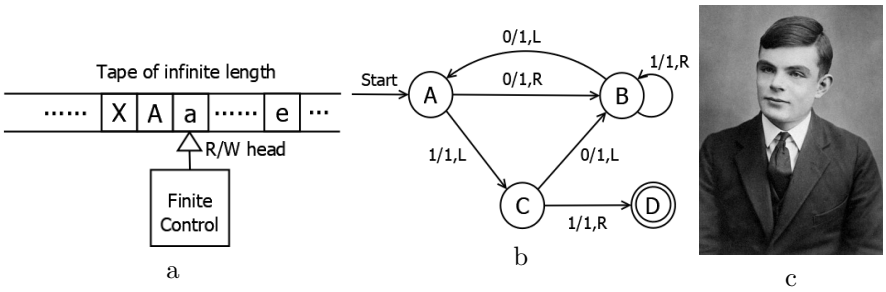

Figure 7: a) Turing machine tape; b) Turing machine state represented as automaton; c) Alan Turing, the machine author.

- A *read/write head* that can read and write symbols on the *tape* and move the *tape* left and right one (and only one) cell at a time.
- A *state register* that stores the state of the Turing machine, one of finitely many. Among these is the special *start state* with which the state register is initialized.
- A finite *table* of instructions that, given the *state* the machine is currently in and the symbol it is reading on the *tape*, tells the machine to do the following in sequence:
  - Either erase or write a symbol.
  - Move the *head* to the left or right cell.
  - Assume the same or a *new state* as prescribed.

After regarding probably most famous automaton, we are now will have a look on several other types of automata. The next thing which we will look into is a linear bounded automaton. It is a Turing Machine that satisfies the following three conditions:

- Its input alphabet includes two special symbols, serving as *left and right endmarkers*.
- Its *transitions* may not print other symbols over the *endmarkers*.
- Its *transitions* may neither move to the left of the *left endmarker* nor to the right of the *right endmarker*.

A Finite State Automaton (FSA), or Finite State Machine (FSM) presented at Fig. 8, consists of:

- A finite number of *states*, while the FSM can be in one *states* at each given time;
- A *head* which read a symbol from a sequence of symbols as the *input*. The *head* always goes to the next symbol at the next time step;

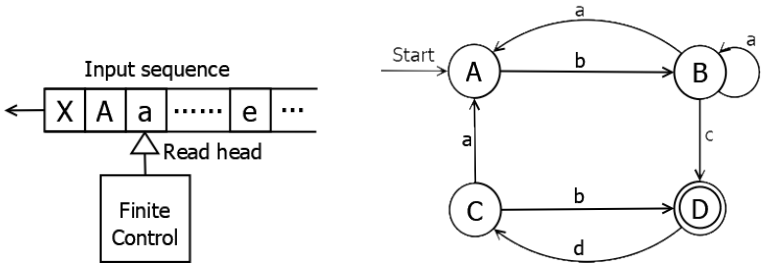

Figure 8: Finite state automaton / machine (FSA/FSM).

- A *transition* matrix which determines the next *states* of the FSM according to the current *states* and the current symbol.

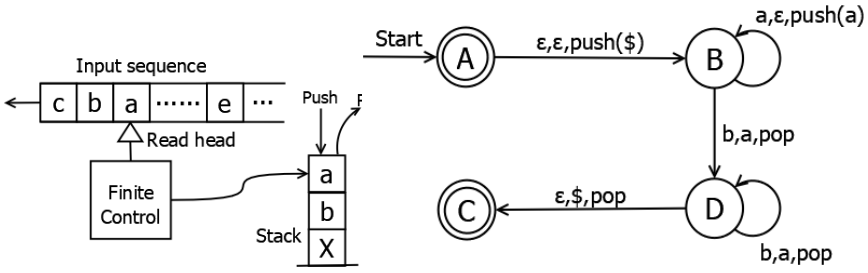

Figure 9: Pushdown automaton (PDA).

The last interesting automaton type is a pushdown Automaton (PDA) presented at Fig. 9. This automaton is similar to FSM except that it maintains a *stack*:

- It can use the top of the *stack* to decide which *transition* to take. In each step, it chooses a *transition* by indexing a table by *input symbol*, *current state*, and the *symbol* at the top of the *stack*.
- It can manipulate the *stack* as part of performing a *transition*. The manipulation can be to *push* a particular symbol to the top of the *stack*, or to *pop* off the top of the *stack*.

4 Text segmentation and morphology analysis

NLP is working with a text, but it is common that a NLP technique is working with smaller units that the texts. In NLP, text is segmented into units of various granularities, which include, but not limited to: Chapters and sections; Paragraphs; Sentences; Clauses; Phrases; Words; Subword units (stems, suffixes, prefixes).

Text segmentation is not straightforward in many cases:

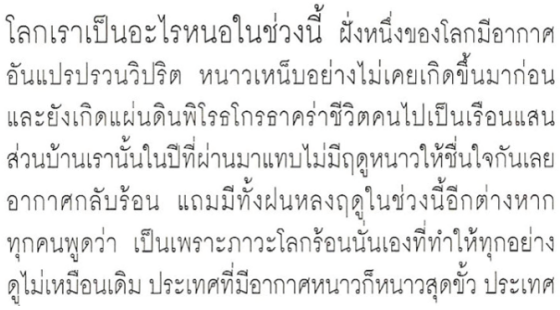

Figure 10: Thai text sample. One cannot rely on the spaces as boundaries between the sentences.

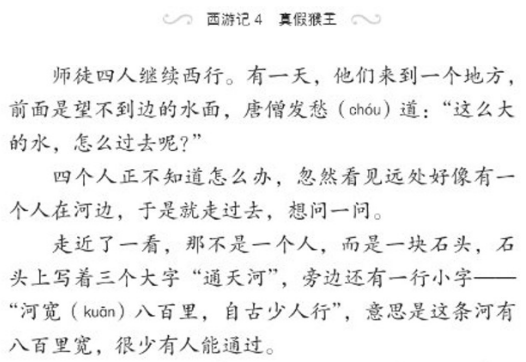

Figure 11: Chinese text sample. There are no spaces between the words.

- For languages like Chinese, Japanese, Tibetan, Thai, there are no spaces between words;
- For languages like Thai and Tibetan, the delimiters between sentences, clauses or phrases are ambiguous, which makes it hard to segment sentences;
- Even for English, sentence segmentation is not a trivial task, because the full stop mark (.) is also used for abbreviations, decimals, etc., which may or may not terminate a sentence.

Some examples from English sentence segmentation. Why the dot marks (.) are ambiguous? The dots could be used as:

- Full stop: *This is an apple.*
- Decimal delimiter: *235.6*
- Integral part of an abbreviation: *U.S. Ph.D. etc.*

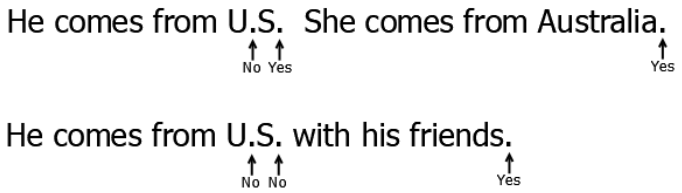

Figure 12: English sentence segmentation.

(a)	下雨天留客天留我不留	Unpunctuated Chinese sentence
	下雨、天留客。天留、我不留！	*It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!*
	下雨天、留客天。留我不？留！	*The rainy day, the staying day. Would you like me to stay? Sure!*
(b)	我喜欢新西兰花	Unsegmented Chinese sentence
	我　喜欢　新西兰　花	*I like New Zealand flowers*
	我　喜欢　新　西兰花	*I like fresh broccoli*

Figure 13: Chinese word segmentation may results in different meanings.

- Sometimes a dot could has multiple roles: *He comes from U.S.*

To segment English text into sentences, we need to determine whether a dot mark is an end of sentence or not. It can be solved as a classification problem. There is an illustrative example of ambiguous case on Fig. 12.

4.1 Morphological analysis

To break a word down into component morphemes and build a structured representation. A morpheme is the minimal meaning-bearing unit in a language. We should notice that a morpheme could not bear a meaning by itself but in some combination with some other morphemes only. The types os the morphemes are:

- **Stem** - the morpheme that forms the central meaning unit in a word. Another name for it is *root*.
- **Affix**. There are several types of affixes. It is simpler to present them by their usage:
  - **Prefix**: e.g., possible → **im**possible
  - **Suffix**: e.g., walk → walk**ing**
  - **Infix**: e.g., hingi → h**um**ingi (Tagalog)
  - **Circumfix**: e.g., sag*en* → **ge**sag**t** (German)

Input: Another **ex-Golden** *Stater,* Paul Stankowski from *Oxnard,* is contending for a berth on the **U.S.** Ryder Cup team after winning his first PGA Tour event last year and staying within three strokes of the lead through three rounds of last *month's* **U.S.** *Open.* **H.J.** Heinz Company said it completed the sale of its **Ore-Ida** frozen-food business catering to the service industry to McCain Foods Ltd. for about *$500* million. *It's* the first group action of its kind in Britain and one of only a handful of lawsuits against tobacco companies outside the U.S.

Output: Another **ex-Golden** *Stater ,* Paul Stankowski from *Oxnard ,* is contending for a berth on the **U.S.** Ryder Cup team after winning his first PGA Tour event last year and staying within three strokes of the lead through three rounds of last *month 's* **U.S.** *Open .* **H.J.** Heinz Company said it completed the sale of its **Ore-Ida** frozen-food business catering to the service industry to McCain Foods Ltd. for about *$ 500* million . *It 's* the first group action of its kind in Britain and one of only a handful of lawsuits against tobacco companies outside the *U.S. .*

Note: *Text in italic*: change, **text in bold**: Keep

Table 3: English word segmentation - Tokenization. An example of Stanford Tokenizer work.

Practically there are two different tasks which are considered to be close. Those are Stemming and Lemmatization. Stemming itself is a process of stem extraction from a word. While the lemmatization is a process of finding a word normal form, which could or could not be the same as its stem. Let us have a look on some examples:

- Stemming:
  - Ex: writing → writ (+ ing)
- Lemmatization:
  - Ex1: writing → write (+V +Prog)
  - Ex2: books → book (+N +Pl)
  - Ex3: writes → write (+V +3Per +Sg)

The stemming example shows the stem (*writ*) and a departed suffix (*ing*). The lemmatization examples show the normal form of the verb "to write", which is *write* (an infinitive form), alongside with some special tags referring to the original word. For the word "writes" these are: *V* - reflecting the fact that this is a verb; *3Per* - the verb in the 3rd person form of a *Sg* singular verb form.

The morphological analysis could be ambiguous. Let us consider an example on Tab. 4.1. There are two possible interpretation of a word "flies" - either it is a plural form of a noun "a fly", or it is a 3rd person singular form of a verb "to fly". Thus ambiguity could be resolved based only on a context.

flies	→	fly +N +PL
flies	→	fly +V +3rd +Sg

Table 4: Example of ambiguous morphology.

Basing on their morphology features languages could be roughly divided into three groups:

- Analytic languages: e.g., Chinese
- Synthetic flexive languages: e.g., Russian
- Synthetic agglutinate languages: e.g., Turkish

These three groups are drastically differ from each other. The analytical languages have (almost) no morphology and use function words to represent a grammatical meaning. The agglutinate languages have in contrast one morpheme for one grammatical feature at once (e.g. grammar gender and grammar number in Spanish word "amigas", where suffix *a* before suffix "s" represents that these are female gender, while *s* itself represents plural form[1]). And the flexive languages are mixing the grammatical meaning as we have seen in "writes" example.

There should be said a few words about English language this notes is written in. This language once was flexive in its history (see Middle English for reference), but the modern English language use morphemes less with time, reducing for example the verb forms to 4 most common ones: infinitive, 3rd person singular form of present tense, past tense, and continuous tense. While in the past there were at least 8 forms for a verb in only one present tense.

There are some ways to combine morphemes to form a word:

- Inflection: stem + gram. morpheme → same class
  - Ex: help + ed → helped
- Derivation: Derivation: stem + gram. morpheme → different class
  - Ex: civil + -zation → civilization
- Compounding: multiple stems
  - Ex: cabdriver, doghouse
- Cliticization: stem + clitic
  - Ex: they**'ll**, she**'s** (*I don't know who she is)

[1]Spanish language itself is not an agglutinate, but a flexive one. This word is chosen as convenient example since it is pretty commonly used.

The compounding usually considered to be a word creation mechanism, rather the morphology one, but if we are using a vocabulary of stems in a morphology parser, we need to consider this case nevertheless. Also clitication is a process of creation a word from the parts of the other words and should be considered due to the same reason.

Once we had considered the morphology, let us return to the automata. There is a class of ones which is widely used to create a morphology parsers. This is the Finite state transducers (FSTs).

- Finite State Transducers are an extension to Finite State Machines, where an output symbol will be given for each input symbol.
- FSTs are commonly used tools for morphological analysis.
- A FST can be used in a inverse direction with the input and the output swapped.

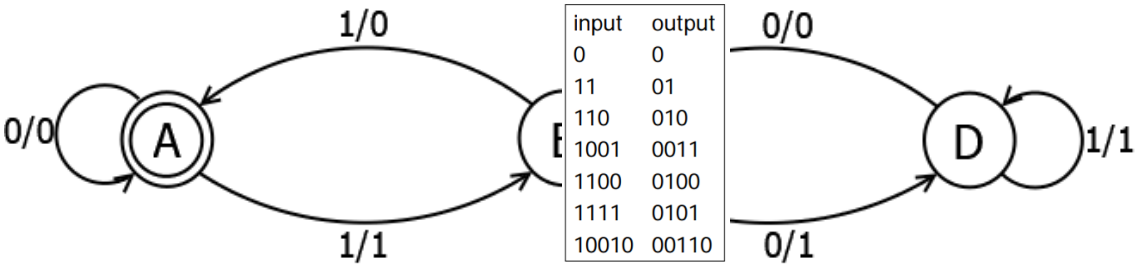

Figure 14: Finite state transducers (FSTs)

Let us consider more closely English morphology. The existing word creation methods in the languages are:

- Affixes: prefixes, suffixes; no infixes, no circumfixes.
- Inflectional:
  - Noun: -s
  - Verbs: -s, -ing, -ed, -ed
  - Adjectives: -er, -est
- Derivational:
  - Ex: V + suf → N
    computerize + -ation → computerization
    kill + er → killer
- Compound: pickup, database, heartbroken, etc.
- Cliticization: 'm, 've, 're, etc.

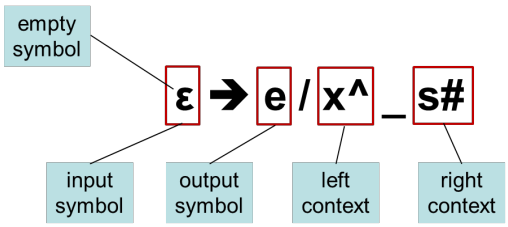

Figure 15: Rewrite rules for the orthography.

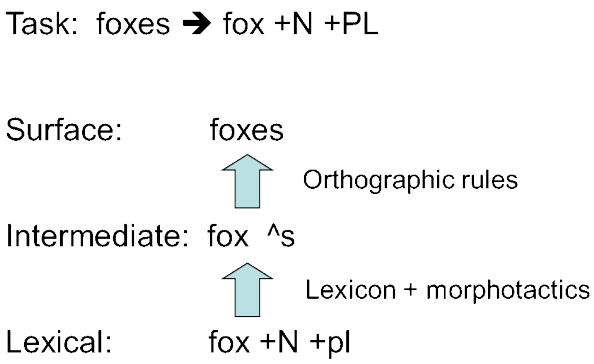

Figure 16: An example of the rewrites rules application.

To make a morphology analyzer for English we need three components:

- Lexicon: the list of stems and affixes, with associated features.
  - Ex1: book: N
  - Ex2: -s: +PL
- Morphotactics:
  - Ex: +PL follows a noun
- Orthographic rules (spelling rules): to handle exceptions that can be dealt with by rules.
  - Ex1: y → ie: fly + -s → flies
  - Ex2: ϵ → e: fox + -s → foxes
  - Ex3: ϵ → e / x ˆ _ s#

The rewrite rule from the Fig. 16 could be presented as a finite-state machine, see Fig. 17. The examples with words "cats, foxes and geese" are presented in Fig. **??**.

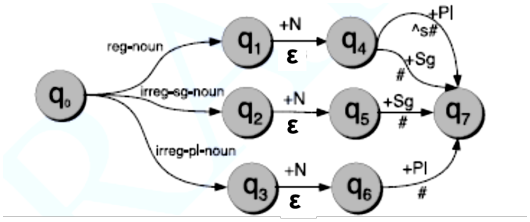

Figure 17: Finite-State Machine to rewrite a word to its plural form.

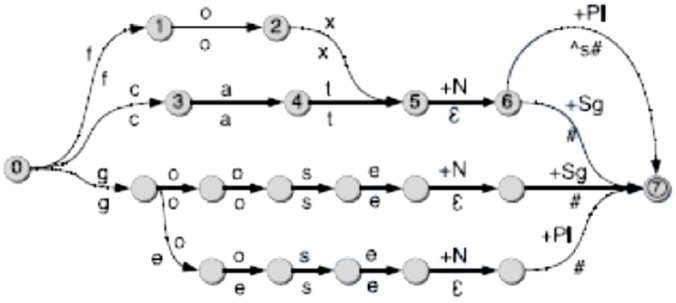

Figure 18: Application of a FST to create plural forms for words "cat, fox, and goose".

5 Word frequency and collocations

Statics from Corpus of the Contemporary American English[2] is presented on the Fig. 19. This figure shows first dozens of 5000 most common words in English. Obviously these words are the most frequent ones, but also they have no specific meaning, like the articles "the" and "a". These not meaningful words usually called as *stopwords* in practice. The stopwords usually are being removed in the preprocessing of a particular NLP task, although there are some exceptions from this rule. But more importantly, we could use the statistics in another way. We could search for a dependency between the number of occurrences of a particular word in a corpus and the number of the words with the same number of occurrences. And there is such dependency, which is described by so called Zipf law after George Kingsley Zipf. George Zipf described this dependency for the English language (see Fig. 20.

The frequency of any word is inversely proportional to its rank in the frequency table. Formally, let:

N be the number of elements; k be their rank; s be the value of the exponent characterizing the distribution. Zipf's law then predicts that out of a population

[2] `http://www.wordfrequency.info/`

Rank	Word	Part of speech	Frequency	Dispersion
1	the	a	22038615	0.98
2	be	v	12545825	0.97
3	and	c	10741073	0.99
4	of	i	10343885	0.97
5	a	a	10144200	0.98
6	in	i	6996437	0.98
7	to	t	6332195	0.98
8	have	v	4303955	0.97
9	to	i	3856916	0.99
10	it	p	3872477	0.96

Figure 19: Top 5000 words in American English

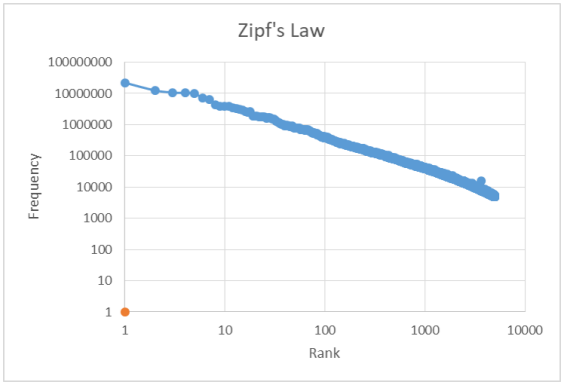

Figure 20: Zipf law for American English

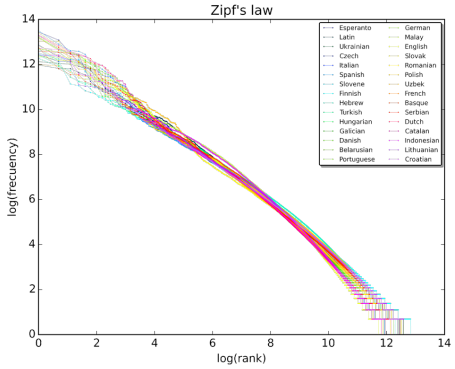

Figure 21: A plot of the rank versus frequency for the first 10 million words in 30 Wikipedias (dumps from October 2015) in a log-log scale. Adopted from Wikipedia.

of N elements, the normalized frequency of elements of rank k, $f(k; s, N)$, is:

$$f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^{N} (1/n^s)}$$

Interestingly, that this law is applicable to the most natural languages, a comparison for 30 languages is presented at Fig. 21.

6 Multi-Word Expressions

A *collocation* is an expression consisting of two or more words that correspond to some conventional way of saying things. The words together can mean more than their sum of parts. These are some examples: The Times of India, disk drive; hot dog, mother in law. The samples are adopted from Manning & Schütze, Fundamentals of Statistical Natural Language Processing, 1999.

These are noun phrases like *strong tea* and *weapons of mass destruction* which could be considered as collocations. The phrasal verbs like to *make up*, and other phrases like the rich and powerful, also could be listed as ones. The collocations usually cannot be translated into other languages word by word. It is worth mentioning, that a phrase can be a collocation even if it is not consecutive (as in the example *knock ... door*).

But not all the multi-word expressions could be considered as collocations. The typical criteria for collocations:

- non-compositionality,
- non-substitutability,

- non-modifiability.

A phrase is *compositional* if the meaning can be predicted from the meaning of the parts, e.g. "new companies". A phrase is *non-compositional* if its meaning cannot be predicted from the meaning of its parts, e.g. "hot dog". Collocations are not necessarily fully compositional in that there is usually an element of meaning added to the combination, e.g. *strong tea*. The idioms are the most extreme examples of non-compositionality, e.g. *to hear it through the grapevine* (to know something from gossip).

We cannot substitute near-synonyms for the components of a collocation. For example, we can't say *yellow wine* instead of *white wine* even though *yellow* is as good a description of the color of *white* wine as white is (it is kind of a yellowish white). Many collocations cannot be freely modified with additional lexical material or through grammatical transformations (*non-modifiability*). Some examples: *white wine*, but not *whiter wine*; *mother in law*, but not *mother in laws*.

There are additional criteria, which could used to determine the collocation: Frequency, Mean and Variance of Distances between Words, Hypothesis Testing, Mutual Information, Left and Right Context Entropy, C-Value.

Additional information could be found in these books and articles:

- Manning & Schütze, Fundamentals of Statistical Natural Language Processing, 1999, Chapter 3 (A general introduction to collocation)
- Katerina T. Frantzi, Sophia Ananiadou, Junichi Tsujii, The C-value / NC-value Method of Automatic Recognition for Multi-word Terms, ECDL 1998: Research and Advanced Technology for Digital Libraries pp 585-604 (proposed the C-value metric)
- Zhiyong Luo, Rou Song, An integrated method for Chinese unknown word extraction, SIGHAN 2004. Barcelona, Spain. (proposed the context entropy method).