Computer and Natural Language

Christoph Leitner

June 4, 2012

1 Introduction

Using search-engines, translating texts automatically, controlling devices using speech commands - all those applications require the processing of natural language by computers. According to Alan Turing and the Turing test, a machine can be considered intelligent, if it can hold a conversation with a human, so that the human doesn't recognize that he is only talking (writing) with a computer program.

The field of *Computational Linguistics* is researching methods to let a computer understand our natural language.

This article provides a brief introduction to some of the basic concepts and techniques of natural language processing.

It is aimed at readers who have a basic knowledge of formal language theory, especially on the *Chomsky hierarchy*, *finite state automata* and *regular languages*.

The article is organized as follows. In section 2 we locate natural languages in the Chomsky Hierarchy of formal languages. In section 3 we focus on the significance of regular languages in natural language processing, introduce regular relations and show their applications in disambiguation.

2 Natural Languages in the Chomsky Hierarchy

It seems quite obvious to use formal grammars to model the structure of natural language. If we manage to build a formal grammar, that generates all and only the sentences of a natural language, we have efficient algorithms to further process the language. For instance: for regular languages we have regular expressions and finite state automata (FSA) to build dictionaries and look up words within these dictionaries. To know which kind of grammar we have to use to describe a natural language, we need to know where that language is located in the Chomsky hierarchy.

Natural languages generally can't be modeled using *only* regular grammars - Noam Chomsky states in a Theorem, that *"English is not a finite state language"*[1] - so no FSA can produce all grammatical sentences of English. However, *some* aspects can be

[1] Noam Chomsky - Syntactic Structures ([1, p. 21])

modeled using regular grammars and *most* aspects can be modeled using context-free grammars.

3 Regular Languages in Natural Language Processing

As mentioned previously, a natural language can not simply be expressed in a regular language. However, lexicons or dictionaries can be built using finite state automata (*FSA*) and regular expressions are useful for parsing input text. Those lexicons are often used for information retrieval needed for spell checking or for part-of-speech tagging. Regular languages are especially interesting for natural language processing because of their simplicity for computation. For instance - recognizing a string using an FSA can be done in linear time in the length of the provided string.

3.1 Finite State Automata, Regular Expressions and ELIZA

In many languages, morphological rules mostly consist of adding affixes (suffixes, prefixes, infixes) to words. In the English language, for instance, the plural form of a word can (mostly) be formed by simply adding the suffix "-s" or "-es" to the word. Therefore, words and affixes can be efficiently stored in a dictionary that uses FSAs (and later mapped by finite state transducers - see 3.3).

A good example for the usage of regular expressions in NLP is the ELIZA program. The ELIZA program was an early natural language processing tool, which simulated a psychologist and could maintain a simple conversation with a user. For these conversations, ELIZA used mostly regular expressions with a memory to perform substitutions within a text - the same way the UNIX program `grep` does.

For example: the user's statement "it seems you like me" would be split into four parts:

1	2	3	4
it seems that	you	like	me

then the decomposition rule `(0 YOU 0 ME)` would be applied and an answer would be formed using the reassembly rule `WHAT MAKES YOU THINK I 3 YOU` (the 0 in the decomposition rule stands for an arbitrary number of words, the 3 in the reassembly rule stands for the third part of the input sentence) leading to the answer: `WHAT MAKES YOU THINK I LIKE YOU`

Even though the rules ELIZA used were quite simple, people interacting with the program often believed that it really understood their problems and refused to believe otherwise, even after they were told that they were just talking to a computer program.

3.2 Regular Relations and Part-of-Speech Tagging

Regular relations - relations over natural languages - are often used to map words of natural language to some useful information regarding that word.

For instance - a simple part-of-speech tagger could map every word of a natural language sentence to their respective part of speech (also called word class).

Consider the two sets $\Sigma_1 = \{a, b, ..., z\}$ and $\Sigma_2 = \{PRON, V, DET, ADJ, N, P\}$[2], where Σ_1 is the alphabet of our natural language and Σ_2 is a set containing part of speech Tags, then

The	DET	of	P
moon	N	green	ADJ
is	V	cheese	N
made	V		

could be pairs in the relation the tagger uses.

3.3 Finite State Transducers

Finite State Transducers (FST) are used to implement relations over two regular languages.

> "DEFINITION: A **finite state transducer** is a six-tuple $(Q, q_0, \Sigma_1, \Sigma_2, \delta, F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, Σ_1 and Σ_2 are alphabets, and δ is a subset of $Q \times \Sigma_1 \times \Sigma_2 \times Q$."[3]

Basically, a FST can be seen as an FSA which generates an output. Each character read from the input may result in a change of state. This transition can output a character. Input and output character are separated by a colon (:).
Example: Figure 1 shows an FST that maps the singular form of a verb to its plural form. It can be used to check whether a input word is a correct plural form. E.g. if the computation ends in state q16 the accepting pair is foot:feet.
Since the same procedure can be applied to a word and a set of tags, a FST can also implement a part-of-speech tagger.

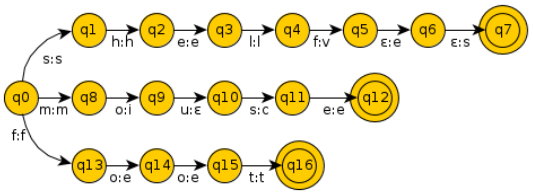

Figure 1: A finite state transducer

3.4 Disambiguation

Ambiguity is a big issue in natural language processing. In natural languages, words can have different meanings, depending on the context they are used in. For example: consider the following two sentences:

1. *Their last song was a great hit.*
2. *She hit him with a frying pan.*

As you can see, in the first sentence, the word *"hit"* is a noun, while it is a verb in the second sentence.

[2] Pronoun, Verb, Determiner, Adjective, Noun, Preposition
[3] [2, p. 25]

If a part-of-speech tagger could assign the correct tag to the *"hit"s* in their respective sentences, the meaning would become clear - the tagger performs a *word-sense disambiguation*.

The problem to solve now is: "How does the tagger know which tag to assign to which word?"

Generally, there are two methods to tell a tagger how to assign tags:

Rule based: a set of rules is applied to fix the assignment of tags

Probabilistic: the tagger assigns the "most probable tag"

Current applications in NLP mostly use probabilistic methods for disambiguation. These methods also make it easier to implement machine learning algorithms, so that the processing machine can acquire the needed probabilities from raw text on their own.

4 Conclusion

In this article, I focused mostly on some foundations of natural language processing. Firstly, I mentioned that the best way to model natural language in a way that computers can process them are formal grammars and that regular grammars are useful for processing natural language, but not expressive enough to cover all aspects.

Secondly, I used the ELIZA program as an example of the applications of FSAs and regex for NLP. I introduced the concept of relations over regular languages and presented the finite state transducer as a way to implement these relations.

Finally, I addressed the problem of ambiguity in NLP and stated the existing methods for disambiguation.

I didn't mention more complicated aspects of NLP like the usage of more expressive grammars (context-free grammars and mildly context-sensitive grammars) or how the mentioned probabilistic methods work. Topics like speech recognition (which also uses probabilistic methods for conversion) were not mentioned either. The interested reader can find more information on these topics in [2] and [3].

References

[1] N. Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

[2] A. Clark, C. Fox, and S. Lappin, editors. *The Handbook of Computational Linguistics and Natural Language Processing*. Blackwell Handbooks in Linguistics. John Wiley & Sons, 2010.

[3] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 1 edition, 2000.

[4] J. Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, 1966.