Natural Language Processing

For us humans, spoken language is the most efficient and convenient way of direct communication.

Therefore, being able to simply speak to a computer instead of using a keyboard or mouse would be desirable.

There are some reasonable approaches to such systems, even though they are still far away from passing the Turing test (see textbook).

Natural Language Processing

Such systems can work surprisingly well as long as they are restricted to a narrow domain.

For example, there are systems that communicate with a hotel's customer to make a room reservation.

The Amtrak reservation system is a good example.

You will find the files eliza.exe and response.dat on the course homepage. Download them into the same folder and start eliza.exe for trying out an (overly) simple approach to natural language communication.

Natural Language Processing

The program ELIZA does not analyze the semantics of the sentences entered by the user.

It just looks for keywords in the sentence and based on those rearranges the words in the sentence to form a question.

When we want to build a system that actually communicates with the user, we need to provide an actual mechanism for analyzing syntax and semantics of the input.

Natural Language Processing

Prior to syntactical analysis, the input has to be scanned to split the sentence into separate words and correct possible spelling mistakes.

It follows the syntactical analysis, called parsing.

After the syntactical analysis, a domain-specific knowledge base is used for semantic analysis.

This allows the system to recognize the information in the user's sentence and perform the appropriate action, e.g., mark a room reservation in the hotel's database.

In the following, we will focus on parsing.

Natural Language Processing

A language is said to be **context-free** if and only if it can be generated by a **context-free grammar**.

Grammars, both for natural and computational languages, are collections of rules defining how to generate strings in that language.

In computer science, these rules are called **productions** or **substitution rules**.

Context-Free Languages

Each production defines how a **nonterminal** (a placeholder) can be replaced by a string consisting of nonterminals and/or **terminals** (symbols of an alphabet).

Notice that these symbols can be anything, also entire **words**.

To produce a string, we take the **start symbol** and apply productions to it in any order until all nonterminals are eliminated.

The set of all possible strings of terminals we can create this way is the **language** specified by the grammar.

Context-Free Languages

Example:

Sentence → Noun-phrase Verb-phrase
Noun-phrase → Noun | Article Noun
Verb-phrase → Verb | Verb Noun-Phrase
Noun → **student** | **professor**
Article → **a** | **the**
Verb → **yawns** | **annoys**

Sentence ⇒ Noun-phrase Verb-phrase
⇒ Article Noun Verb Noun-phrase
⇒ **the professor annoys** Article Noun
⇒ **the professor annoys a student**

Context-Free Languages

Why are such grammars called “context-free” ?

Because the productions can be applied regardless of the context – the neighboring symbols – of the nonterminal to be replaced.

Let us take a look at the following context-free grammar with the following productions:

A → xy

A → xAy

which can be combined in a single production:

A → xAy | xy

Context-Free Languages

Which words can we generate with this grammar, that is, with the rule

A → xAy | xy ?

A ⇒ xy
A ⇒ xAy ⇒ xxyy
A ⇒ x**A**y ⇒ x**xAy**y ⇒ xxxyyy
A ⇒ xAy ⇒ xx**A**yy ⇒ xx**xAy**yy ⇒ xxxxyyyy

L(G) = {xy, xxyy, xxxyyy, …} = $\{x^{[n]}y^{[n]} \mid n > 0\}$.

For natural language processing, we run this process in the opposite direction (parsing).

Parsing

You remember the substitution rules?

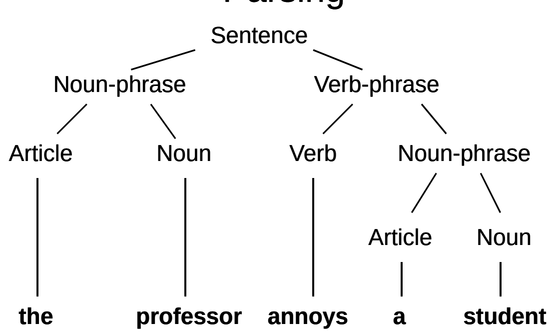
Sentence → Noun-phrase Verb-phrase
Noun-phrase → Noun | Article Noun
Verb-phrase → Verb | Verb Noun-Phrase
Noun → **student** | **professor**
Article → **a** | **the**
Verb → **yawns** | **annoys**

For natural language processing, we apply these rules in the opposite direction (**parsing**).

We start with a complete sentence and reduce it step-by-step until only the “Sentence” symbol is left.

This way we record a **parse tree** that tells us the syntactical structure of the sentence.

Parsing



Parsing

Let us take a look at how parsing can be done using a simpler example:

A → xAy | xy

xy ⇒ A

xxyy ⇒ xAy ⇒ A

xxxyyy ⇒ x**xAy**y ⇒ x**A**y ⇒ A

xxxxyyyy ⇒ xx**xAy**yy ⇒ xx**A**yy ⇒ xAy ⇒ A

This way we can check whether an input is in our language, and at the same time we can build the parse tree.

Expert Systems

From the very beginnings of computing, computers were used to perform **numerical calculations**, e.g., for construction or cryptography.

Obviously, computers clearly **outperform** humans in doing such tasks.

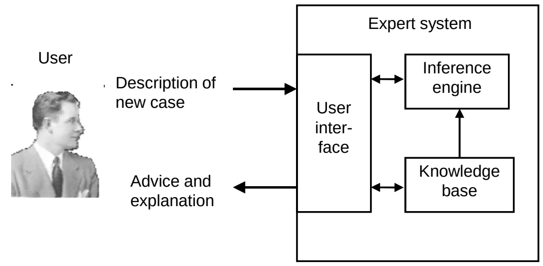
However, it turned out to be possible and useful to write computer programs that process **high-level knowledge**.

These systems incorporate **domain-specific expert knowledge** and can apply this knowledge to support decision making in particular cases.

We call them **expert systems**.

Expert Systems

A simple sketch of an expert system:



Expert Systems

The knowledge base usually contains a large set of **rules**.

The inference engine uses inference rules to evaluate the information entered by the user **(remember?)**.

Typically, it will dynamically generate a **series of questions** for the user.

Often, the choice of the next question will depend on the user's previous answers.

After giving its **diagnosis and advice**, the expert system has to **explain** to the user how it came to its conclusions.

Expert Systems

This way expert systems can support decision making by "other" experts.

Example: The system MYCIN assists doctors in choosing antibiotics for patients with bacterial infections.

Sample rule:

If (i) the infection is meningitis

and (ii) organisms were not seen in the stain of the culture

and (iii) the type of infection may be bacterial

and (iv) the patient has been seriously burned

then there is suggestive evidence that Pseudomonas aeruginosa is one of the organisms that might be causing the infection.

Expert Systems

The simplest way of building expert systems is based on **decision trees**.

A (mathematical) tree consists of **nodes**, one of them called the **root**, and **branches** that connect the nodes.

Nodes that are connected by only a single branch are called **leaves**, the others are called **inner nodes**.

In a **decision tree**, each inner node corresponds to a **question** and has an outgoing branch for each possible **answer/decision**.

The leaves of the decision tree indicate the possible **outcomes** of making a specific series of decisions.

Expert Systems

On the course homepage you will find the program **expert.exe** and a sample tree file **sample.txt**.

Please download both files to a PC into the same directory and start expert.exe.

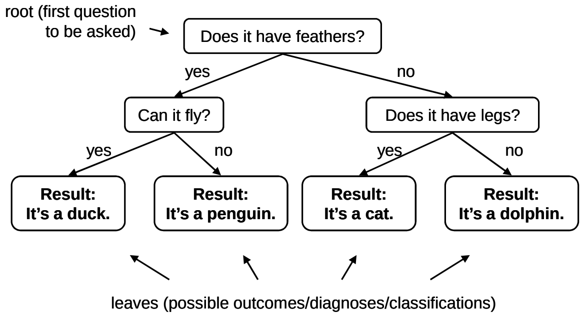
Tell it that you want to load a file and then enter the name sample.txt.

Choose menu option 2 to start the system.

You can try out all menu options to build, modify, and save your own expert systems.

Of course this program is very simple – it only uses **yes/no questions** to build a **binary decision tree.**

Example: Identification of Species



Example: Identification of Species

Now imagine that you are seeing a **dog** and, since you do not know what a dog is, want the system to identify this species.

The system will actually tell you that it is a **cat**.

However, you are sure that it is not a cat, and so you consult an expert.

The expert identifies it as a dog and tells you that the easiest way to tell a dog from a cat is that dogs **bark**.

Expert.exe will ask you to provide the **new species** and the **question** whose answer differentiates the new species from the system's previously chosen one.

It will **expand** the binary decision tree as follows:

Example: Identification of Species