Computational Linguistics: Introduction

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

P.zza Domenicani 3, Room: 2.01, e-mail:

bernardi@inf.unibz.it

Contents

1. Course Info

- ▶ **Time**: Thursdays 10:30-12:30 (Lessons) and Wednesdays: 18:00-19:00 (Labs)

- ▶ **Office hours**: Oct-Jan: Thursdays 13:00-14:00, later by prior arrangement via e-mail.

- ▶ **Course Materials**: Slides, Readings (**study the book!**)

- ▶ **Reference Material**:

  1. D. Jurasfky and J. H. Martin **Speech and Language Processing**. (see nr. of chapters on the web –old edition.).

  2. P. Blackburn and J. Bos (BB1) (see nr. of chapters on the web) **Representation and Inference for Natural Language A First Course in Computational Semantics**

  3. P. Blackburn and J. Bos (BB2) (see nr. of chapters on the web) **Working with Discourse Representation Structures**

  4. P. Blackburn and K. Striegnitz (BS) (online) **Natural Language Processing Techniques in Prolog**

  5. P. Blackburn, J. Bos and K. Striegnitz. **Learn Prolog Now!**.

- ▶ **Url**: http://www.inf.unibz.it/~bernardi/Courses/CompLing/10-11.html

1.1. Grading

1. **Projects**: You are to complete an independent critique plu project on some topic in CL that must include a careful write-up or oral presentation (overview of the literature, a critique of a selected paper and a description of your own idea/implementation). [Elena Cabrio] [50%]

2. **Final Exam**: Written exercises on the topics discussed in class. [RB] [50%]

Calendar Last lecture: Jan 13th. Last Lab (project presentation): Jan 20th (TBC). Final exam: Feb 19th (TBC).

Hours of your work This is a 4 ECTS (100 hs): **36 hs** with me and Elena, **64 hs** on your own.

1.2. Program

- **Bernardi**: Fundamentals of Linguistics and Computational Lingusitics with emphasis on: Logic & Language focus on Syntax, Semantics and their interface.
- **Cabrio**: Discussions about some cutting-edge topics.

2. Goals of Computational Linguistics

▶ **Ultimate goal**: To build computer systems that perform as well at using natural language as humans do.

▶ **Immediate goal** To build computer systems that can process text and speech more intelligently.
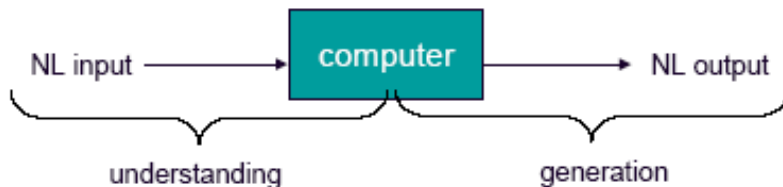
where, NL (**Natural Language**) is the language that people use to communicate with one another and **process** means to analyze.

Unstructured versus Structured Information:
What does it mean?

Structured Information: *Semantics of information captured in DB schema*

Name	Occupation	Organization	Age	Office Location
Jones	Engineer	IBM	29	San Francisco
Smith	Journalist	HP	32	Palo Alto
Burns	President	Xerox	48	Rochester

Unstructured Information: *Semantics inherent in usage and context*

Ursula M Burns has been appointed Xerox's new president in Rochester, today.
Burns, 48, was previously president of the company's business group operations, responsible for global research, engineering, marketing and manufacturing of Xerox technology, supplies and related services.

Xerox

The discovery of semantics in unstructured information & their *explicit representation in* structure *through automated Text (or multi-modal) Analysis*

Located-in

Organization

Name

Ursula M Burns has been appointed Xerox's new president in Rochester, today.
Burns, 48, was previously president of the company's business group operations, responsible for global research, engineering, marketing and manufacturing of Xerox technology, supplies and related services.
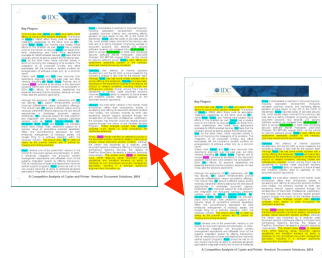
Time

Age

Role
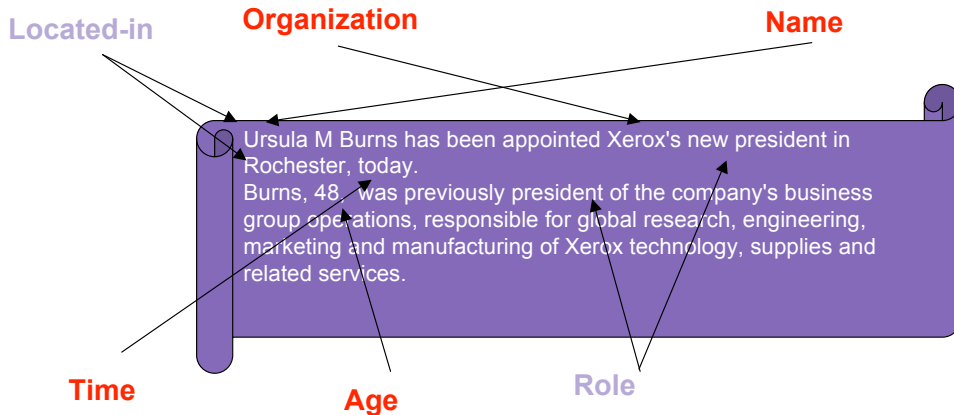
Analysis Engines discover Semantics & build in structure allowing us to process text as we would structured data.

Semantic Analysis of Internet Content

Find document parts, not only entire documents
Focus on the relevant parts of a document
Extract common information from multiple documents

Since, the gist is in the textual data crunching, we need to develop methods to refine what is extracted from documents. Fact extraction to get answers from the web (from syntactic to semantic queries)

Who?

What?

When?

Where?

How?

How many people does Xerox employ?

What is new in image processing?

Capitalise on the extracted knowledge in order to analyse it (inference)

No standard technologies exist to carry out this task.

3. Project at FUB: CACAO

4. Project at FUB: BoB

Links:

CACAO: `http://cacao.inf.unibz.it/cacaoUI/`

MuSiL: `http://pro.unibz.it/opacdocdigger/index.asp?`

BoB: `http://web.inf.unibz.it/bob/`

5. Project at FUB: MiVaS
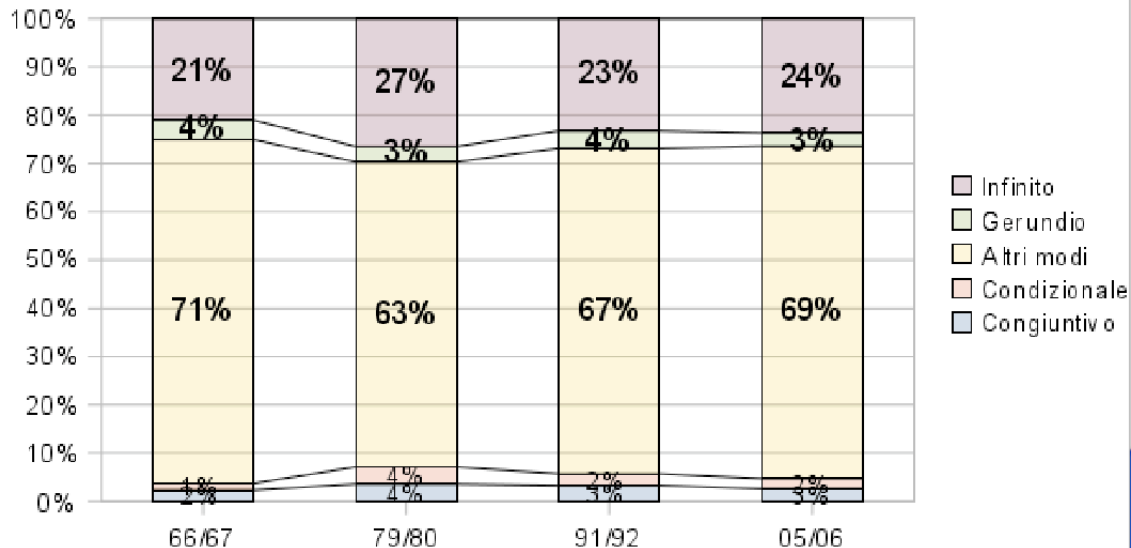
6. Project at FUB: Logic & Language

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{s_4 \vdash s_6 \mid \quad \mid s_7 \vdash s_0}{(s_4 \oslash s_0) \vdash (s_6 \oslash s_7) \mid} \oslash}{((s_6 \oslash s_7) \oslash s_4) \vdash s_0 \mid \quad \mid s_1 \vdash s_9} \Rightarrow}{(((s_6 \oslash s_7) \oslash s_4) \oslash s_9) \vdash (s_0 \oslash s_1) \mid} \oslash}{\cfrac{np_2 \vdash np_3 \mid \quad \mid s_4 \vdash ((s_6 \oslash s_7) \oplus ((s_0 \oslash s_1) \oplus s_9))}{\mid (np_3\backslash s_4) \vdash (np_2\backslash ((s_6 \oslash s_7) \oplus ((s_0 \oslash s_1) \oplus s_9)))} \rightleftharpoons \backslash} \quad np_8 \vdash np_5 \mid}{\mid ((np_3\backslash s_4)/np_5) \vdash ((np_2\backslash ((s_6 \oslash s_7) \oplus ((s_0 \oslash s_1) \oplus s_9)))/np_8)} / }{\mid np_8 \vdash (((np_3\backslash s_4)/np_5)\backslash (np_2\backslash ((s_6 \oslash s_7) \oplus ((s_0 \oslash s_1) \oplus s_9)))) } \Leftarrow}{\mid ((s_6 \oslash s_7) \oslash np_8) \vdash (((np_3\backslash s_4)/np_5)\backslash (np_2\backslash ((s_0 \oslash s_1) \oplus s_9)))} \oslash}{\mid np_2 \vdash (((s_0 \oslash s_1) \oplus s_9)/(((np_3\backslash s_4)/np_5) \otimes ((s_6 \oslash s_7) \oslash np_8)))} \Leftarrow}{\mid ((s_0 \oslash s_1) \oslash np_2) \vdash (s_9/(((np_3\backslash s_4)/np_5) \otimes ((s_6 \oslash s_7) \oslash np_8)))} \oslash$$

$$(((s_0 \oslash s_1) \oslash np_2) \otimes (((np_3\backslash s_4)/np_5) \otimes ((s_6 \oslash s_7) \oslash np_8))) \vdash s_9 \mid \rightleftharpoons$$

$$\mu\alpha_0.\langle \textbf{ everyone } \mid \beta_0 \oslash (\widetilde{\mu}z_0.\langle \textbf{ someone } \mid \alpha_1 \oslash (\widetilde{\mu}y_1.\langle \textbf{ teases } \mid ((z_0 \times \widetilde{\mu}z_1.\langle (\mu\alpha_2.\langle (z_1 \succ \alpha_2) \mid \alpha_1 \rangle \succ \alpha_0) \mid \beta_0 \rangle) \bowtie y_1) \rangle) \rangle) \rangle$$

$$\lambda x_0.(\textbf{everyone } \lambda y_0.(\lambda z_0.(\textbf{someone } \lambda x_1.(\textbf{teases } \lambda x_4.((x_4 \; \lambda z_1.(y_0 \; \lambda x_5.(x_1 \; \lambda x_5.((z_5 \; (x_5 \; x_0)) \; z_1)))) \; z_0)))))$$

$$\lambda x_0.(\mathcal{A}_{y_7}^{x_7} \; (\mathcal{E}_{y_9}^{x_9} \; (x_0 \; (y_7 \; (y_9 \; ((\textbf{teases } x_9) \; x_7))))))$$

7. Project at FUB: Vector Semantics Models

	runs	legs
dog	1	4
cat	1	5
car	4	0
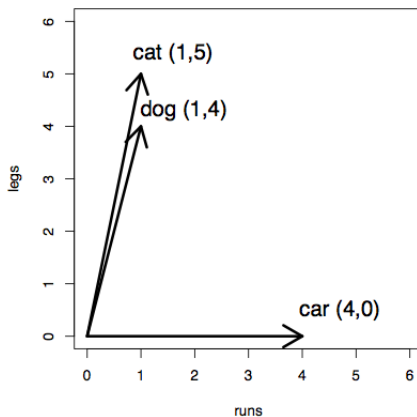
8. The study of Natural Language

Natural Language is studied in several different academic disciplines, and each of them has its set of problems and tools.

Discipline	Typical Problems	Tools
Linguistics	How do words form phrases and sentences? What constrains the possible meanings for a sentence?	Intuitions about well formedness and meaning; mathematical models of structure
Psycoling.	How do people identify the sentence structures? How are word meanings identified? When does understanding take place?	Experimental techniques based on measuring human performance; statistical analysis of observations
Philosophy	What is meaning? How do words and sentences acquire it? How do words identify objects in the world?	Natural language argumentation using intuition about counter-examples; math. models (eg. logic and model theo
Com. Ling.	How is the structure of sentences identified? How can knowledge and reasoning be modeled? How can language be used to accomplish specific tasks?	Algorithms, data structures; formal models of representation and reasoning; AI techniques; math. models

9. Why computational models of NL

There are two motivations for developing computational models:

► **Scientific**: To obtain a **better understanding** of how language works. Computational models may provide very specific predictions about human behavior that can then be explored by the phsycholinguist.

► **Technological**: natural language processing capabilities would revolutionize the way computers are used. Computers that could understand natural language could access to all human knowledge. Moreover, **natural language interfaces** to computers would allow complex systems to be accessible to everyone. In this case, it does not matter if the model used reflects the way humans process language. It only matters that it works.

We are interested in **linguistically motivated computational models** of language understanding and production that can be shown to perform well in specific example domains.

10.1.   Ambiguity: Phonology

Phonology: It concerns how words are related to the sounds that realize them. It's important for speech-based systems.

1. "I scream"

2. "ice cream"

10.2.   Ambiguity: Morphology

Morphology: It's about the inner structure of words. It concerns how words are built up from smaller meaning-bearing units.

1. Unionized (characterized by the presence of labor unions)

2. un-ionized in chemistry

10.3. Ambiguity: Syntax

Syntax: It concerns sentence structure. Different syntactic structure implies different interpretation.

1. I saw the man with the telescope

   ▶ [I[[saw]$_v$[the man]$_{np}$[with the telescope]$_{pp}$]$_{vp}$]$_s$ [(I have the telescope)]
   ▶ [I[[saw]$_v$[[the man]$_{np}$[with the telescope]$_{pp}$]$_{np}$]$_{vp}$]$_s$ [(the man has the telescope)]

2. Visiting relatives can be tiring.

10.4. Ambiguity: Semantics

Semantics: It concerns what words mean and how these meanings combine to form sentence meanings.

1. Visiting relatives can be tiring.
2. Visiting museums can be tiring.

Same set of possible syntactic structures for this sentence. But the meaning of museums makes only one of them plausible.

10.5. Ambiguity: Discourse

Discourse: It concerns how the immediately preceding sentences affect the interpretation of the next sentence

1. **Merck & Co.** formed **a joint venture** with **Ache Group**, of Brazil. **It** will ...?

2. Merck & Co. formed a joint venture with Ache Group, of Brazil. **It**$_i$ will be called Prodome Ltd.

   (**a joint venture!**$_i$)

3. Merck & Co. formed a joint venture with Ache Group, of Brazil. **It**$_i$ will own 50% of the new company to be called Prodome Ltd.

   (**Merck & Co.**$_i$!)

4. Merck & Co. formed a joint venture with Ache Group, of Brazil. **It**$_i$ had previously teamed up with Merck in two unsuccessful pharmaceutical ventures.

   (**Ache Group**$_i$!)

11. Challenges (Slide by F. Segond (Xerox))

The difficulty

can't, $22.50, New York, so-called, a priori
Buy, bought, bought, cat, cats

The old train...
("(The old train) left (the station)")
vs ("(The old)train (the young)").

I know more beautiful women than Julia Roberts.
"I know women more beautiful than Julia Roberts" or "I know more beautiful women than Julia Roberts does".

The boy sees the girl on the hill with the telescope

How to split sentences into words?

How to link words together?

Who knows what?

How to make sense of a phrase/how group words together?

actor

Mary takes a glass of wine

Mary takes a gym course

Mary takes a husband

experience

?

12. Challenges: (ditto)

The difficulty

No! Kids are good at cooking snacks

KIDS MAKE NUTRITIOUS SNACKS

You had kids for appetizers?

No! they use powdered milk

Really? They transformed into powder???

MILK DRINKERS ARE TURNING TO POWDER

No! The boat was repaired

Oh my God! Did she go to the hospital?

QUEEN MARY HAVING BOTTOM SCRAPED

13. NLP Systems

1. Tokenization (Magnini's course)

2. PoS tagging (Magini's course)

3. Morphological analysis (Pianta's course)

4. Shallow parsing

5. Deep parsing

6. Semantic representation (of sentences)

7. Discourse representation

Tokenization It consists in dividing the sequence of symbols in minimum units called tokens (words, date, numbers, punctuation etc..). Many difficulties: e.g.

Sig. Rossi vs. 05.10.05 vs. www.unibz.it;

given up (multi words 1 token).

14. Words: Classes

Traditionally, linguists classify words into different categories:

▶ **Categories**: words are said to belong to *classes*/categories. The main categories are nouns (n), verbs (v), adjectives (adj), articles (art) and adverbs (adv).

The class of words can be divided into two broad supercategories:

1. **Closed Class**: Those that have relatively fixed membership. E.g. prepositions, pronouns, particles, quantifiers, coordination, articles.

2. **Open Class**: nouns, verbs, adjectives, adverbs.

15. Words: Classes (Cont'd)

A word in any of the four open classes can be used to form the basis for a phrase. This word is called the **head** of the phrase and indicates the type of thing, activity, or quality that the phrase describes. E.g. "dog" is the head in: "The dog", "the small dog", "the small dog that I saw".

▶ **Constituents**: Groups of categories may form a single *unit or phrase* called constituents. The main phrases are noun phrases (np), verb phrases (vp), prepositional phrases (pp). Noun phrases for instance are: "she"; "Michael"; "Rajeev Goré"; "the house"; "a young two-year child".

Tests like substitution help decide whether words form constituents.

Can you think of another test?

See Jurafsky & Martin, pp. 289-296 for more details on the single categories and phrases.

15.1. Applications of PoS tagging

More recently, linguists have defined classes of words, called **Part-of-Speech** (PoS) tagsets with much larger numbers of word classes. PoS are used to label words in a given collection of written texts (Corpus). These labels turn out to be useful in several language processing applications.

► **Speech synthesis**: A word's PoS can tell us something about how the word is pronounced. E.g. "content" can be a noun or an adjective, and it's pronounced differently: CONtent (noun) vs. conTENT (adjective).

► **Information Retrieval**: A word's PoS can tell us which morphological affixes it can take, or it can help selecting out nouns or other important words from a document.

► **Theoretical Linguistics**: Words' PoS can help finding instances or frequencies of particular constructions in large corpora.

PoS tagging techniques is a topic of the "Text Processing" course (FBK-irst: Bernardo Magnini)

16. IR plurals vs. singular

I searched for the plural form (foxes) and found pages with the singular. Is it always so trivial?

What about if I search for an irregular noun e.g. sg. "goose" pl. "geese"?

We need to search for the "stem" (or root), i.e. we need a "stemmatizer"

17. Morphology

Morphology is the study of how words are built up from smaller meaning-bearing units, morphemes. It concerns the inner structure of words.

For instance,

▶ **fog**: it's one morphem

▶ **cats**: it consists of two morphemes: cat + -s.

17.1.   Morphemes

Morphemes are divided into:

1. **stems**: they are the main morpheme of the word, supplying the main meaning.

2. **affixes**: they add additional meanings of various kinds. They are further divided into:

   ▶ **prefixes**: precede the stem (English: unknown= un + known)
   ▶ **suffixes**: follow the stem (English: eats= eat + -s)
   ▶ **circumfixes**: do both (German: gesagt (said)= ge + sag + t)
   ▶ **infixes**: are inserted inside the stem (Bontoc -Philippines - fikas (strong), fumikas (to be strong))

A word can have more than one affixes (e.g. re+write+s, unbelievably= believe (stem), un-, -able, -ly).

17.2. Ways of forming new words

There are two basic ways used to form new words:

1. **Inflectional forms**: It is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the **same class as the original stem**, and usually filling some syntactic function like agreement. E.g. in English,

   past tense on verbs is marked by the suffix "-ed", form by "-s", and participle by "-ing".

2. **Derivational forms**: It is the combination of a word stem with a grammatical morpheme, usually resulting in a word of a **different class**, often with a meaning hard to predict exactly. E.g.

   Adverbs from noun: friendly from friend.

   Noun from verbs: killer from kill.

   Adjectives from nouns: "computational" from "computation", "unreal" from "real".

18. Computational Morphology

We want to build a system able to provide the stem and the affixes given a word as input (e.g. cats → {cat + N + PL}), or able to generate all the possible words made of a given stem (e.g. cat → {cats, cat}). To this end, we first of all need to have a way to formally represent Morphology Theory studied by Linguists.

18.1. Modules

To build a morphological recognizer/generator, we'll need at least the following:

lexicon: the list of stems and affixes, together with basic information about them (e.g. Noun stem or Verb stem).

morphotactics: the model of the morpheme ordering, e.g. English plural morpheme follows the noun rather than preceding it.

orthographic rules: spelling rules used to model the changes that occur in a word, e.g. city becomes cities, i.e. "y" ⤳ "ie".

18.2. The Lexicon and Morphotactics

Lexicon: It's a repository of words. Having an explicit list of every word is impossible, hence the lexicon is structured with a list of each of the stems and affixes of the language.

Morphotactics: One of the most common way to model morphotactics is by means of **Finite State Automata** (FSA).

19. Background Notions

Before looking at how FSA are used to recognize/generate natural language morphology we need to introduce some background notions, namely **Formal Languages** and FSA.

Remark: The topics of this section are treated in details in Calvanese's course **Formal Languages**.

19.1.   In short: Formal Language & FSA

▶ A formal language is a set of strings. E.g. $\{a, b, c\}$, $\{the, a, student, students\}$.

▶ Strings are by definition finite in length.

▶ The language accepted (or recognized) by an FSA is the set of all strings it recognizes when used in recognition mode.

▶ The language generated by an FSA is the set of all strings it can generate when used in generation mode.

▶ The language accepted and the language generated by an FSA are exactly the same.

▶ FSA recognize/generate Regular Language.

20.   FSA for Morphology Recognition/Generation

We have said that a language is a set of strings. An important operation on strings is concatenation.

▶ At syntactic level, strings are words that are concatenated together to form phrases.

▶ At morphological level, strings are morphemes that are concatenated to form words. E.g.

$$\text{Stem Language:} \quad \{work, talk, walk\}.$$
$$\text{Suffix Language:} \quad \{\epsilon, -ed, -ing, -s\}.$$

The concatenation of the Suffix language after the Stem language, gives:

$\{work, worked, working, works, talk, talked, talking, talks,$
$walk, walked, walking, walks\}$

20.1. FSA for English Inflectional Morphology

Let's build an FSA that recognizes English nominal inflection. Our lexicon is:

reg-stem	plural	pl-irreg-stem	sing-irreg-stem
fox	-s	geese	goose
cat		sheep	sheep
dog		mice	mouse
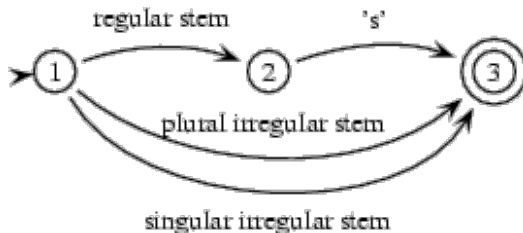
20.2. FSA for English Derivational Morphology

Let's build an FSA that recognizes English adjectives. Our lexicon is:

adj-root1	adj-root2	Suffix-1-2	Suffix-1	Affix-1
clear	big	-er	-ly	un-
happy	cool	-est		
real				
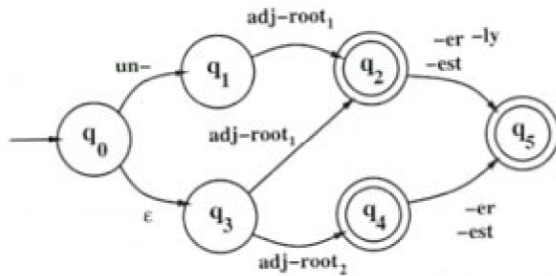
21. Recognizers vs. Parsers

We have seen that we can give a word to a recognizer and the recognizer will say "yes" or "no". But often that's not enough: in addition to knowing that something is accepted by a certain FSA, we would like to have an explanation of why it was accepted. Finite State Parsers give us that kind of explanation by returning the sequence of transitions that was made.

This distinction between recognizers and parsers is a standard one:

▶ **Recognizers** just say "yes" or "no", while

▶ **Parsers** also give an analysis of the input (e.g. a parse tree).

This distinction does not only apply to FSA, but also to all kinds of machines that check whether some input belongs to a language and we will make use of it throughout the course.

21.1. Morphological Parsers

The goal of morphological parsing is to find out what morphemes a given word is built from. For example, a morphological parser should be able to tell us that the word "cats" is the plural form of the noun stem "cat", and that the word "mice" is the plural form of the noun stem "mouse". So, given the string "cats" as input, a morphological parser should produce an output that looks similar to {cat N PL}.

22.   What are FSA good for in CL?

Finite-state techniques are widely used today in both research and industry for natural-language processing. The software implementations and documentation are improving steadily, and they are increasingly available. In CL they are mostly "lower-level" natural language processing:

- ▶ Tokenization
- ▶ Spelling checking/correction
- ▶ Phonology
- ▶ Morphological Analysis/Generation
- ▶ Part-of-Speech Tagging
- ▶ "Shallow" Syntactic Parsing

Finite-state techniques cannot do everything; but for tasks where they do apply, they are extremely attractive. In fact, the flip side of their **expressive weakness** being that they usually behave very well computationally. If you can find a solution based on finite state methods, your implementation will probably be **efficient**.

23.   Practical Info

▶ Labs:

1. Some paper-and-pencile exercises on Syntax (CFG, CG) and Semantics (lambda calculus).

2. Exercises with Prolog – depending on your background

3. Reading groups with Elena

▶ Information Sheet, please fill it in and give it to me now.

24.   Finite State Automata

A finite state **generator** is a simple computing machine that **outputs** a sequence of symbols.

It starts in some **initial state** and then tries to reach a **final state** by making transitions from one state to another.
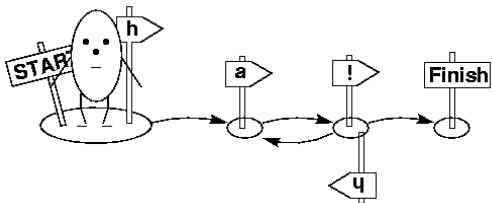
Every time it makes such a transition it emits (or writes or generates) a symbol.
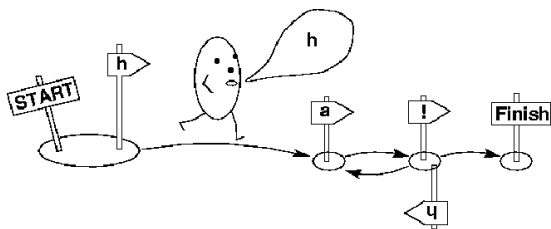

It has to keep doing this until it reaches a **final state**; before that it cannot stop.

So, what does the generator in the pictures say?

It laughs: It generates sequences of symbols of the form `ha!` or `haha!` or `hahaha!` or `hahahaha!` and so on.

Why does it behave like that? Well, it first has to make a transition emitting `h`. The state that it reaches through this transition is not a final state. So, it has to keep on going emitting an `a`. Here, it has two possibilities: it can either follow the `!` arrow, emitting `!` and then stopping in the final state or it can follow the `h` arrow emitting an `h` and going back to the state where it just came from.

24.0.1. FSA as directed graph Finite state generators can be thought of as **directed graphs**. And in fact finite state generators are usually drawn as directed graphs. Here is our laughing machine as we will from now on draw finite state generators:
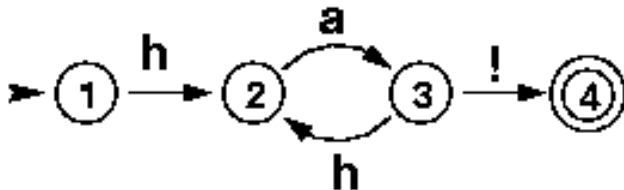
The nodes of the graph are the states of the generator. We have numbered them, so that it is easier to talk about them. The **arcs** of the graph are the transitions, and the **labels** of the arcs are the symbols that the machine emits. A double circle indicates that this state is a final state and the one with the black triangle is the start.

24.0.2. Finite State Recognizer

Finite state recognizers are simple computing machines that **read** (or at least try to read) **a sequence of symbols** from an input tape. That seems to be only a small difference, and in fact, finite state generators and finite state recognizers are exactly the same kind of machine. Just that we are using them to output symbols in one case and to read symbols in the other case.

An FSA recognizes (or accepts) a string of symbols if starting in an intial state it can read in the symbols one after the other while making transitions from one state to another such that the transition reading in the last symbol takes the machine into a final state.

That means an FSA **fails** to recognize a string if:

▶ it cannot reach a final state; or

▶ it can reach a final state, but when it does there are still unread symbols left over.

24.0.3. Recognizer: an example So, this machine recognizes a laughter.
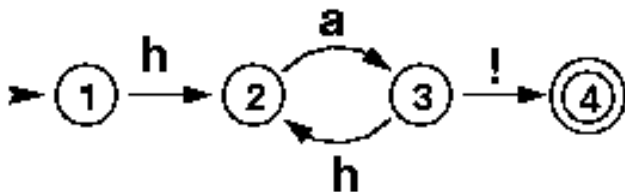

For example, it accepts the word `ha!` by going from state 1 via state 2 and state 3 to state 4. At that point it has read all of the input and is in a final state. It also accepts the word `haha!` by making the following sequence of transitions: state 1, state 2, state 3, state 2, state 3, state 4. Similarly, it accepts `hahaha!` and `hahahaha!` and so on. However, it does not accept the word `haha?`. Although it will be able to read the whole input (state 1, state 2, state 3, state 2, state 3), it will end in a non-final state without anything left to read that could take it into the final state. So, when used in recognition mode, this machine **recognizes exactly the same words that it generates**, when used in generation mode. This is something which is true for all finite state automata.

24.0.4. Finite State Automata Try to think of what language is recognized or generated by the FSA below.
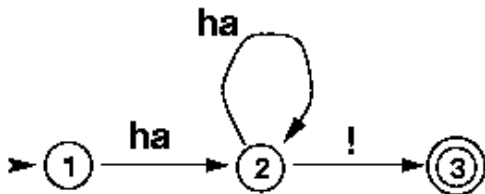
24.0.5. Finite State Automata with jumps
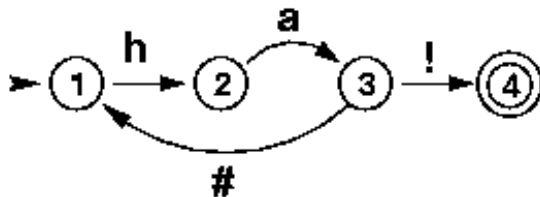
It has a strange transition from state 3 to state 1 which is reading/emitting #. We will call transitions of this type **jump arcs** (or ϵ transitions). Jump arcs let us jump from one state to another **without emitting or reading a symbol**. So, # is really just there to indicate that this is a jump arc and the machine is not reading or writing anything when making this transition.

This FSA accepts/generates the same language as our first laughing machine, namely sequences of `ha` followed by a `!`. Try it yourself.

24.0.6. Important properties of FSA

▶ All in all, finite state generators can only have a **finite number** of different **states**, that's where the name comes from.

▶ Another important property of finite state generators is that **they only know the state they are currently in**. That means they cannot look ahead at the states that come and also don't have any memory of the states they have been in before or the symbols that they have emitted.

▶ An FSA can have several intial and final states (it must have at least one initial and one final state, though).

25. Regular Language

Recall: V^* denotes the set of all strings formed over the alphabet V. A^* denotes the set of all strings obtained by concatenating strings in A in all possible ways.

Given an alphabet V,

1. $\{\}$ is a regular language

2. For any string $x \in V^*$, $\{x\}$ is a regular language.

3. If A and B are regular languages, so is $A \cup B$.

4. If A and B are regular languages, so is AB.

5. If A is a regular language, so is A^*.

6. Nothing else is a regular language.

Examples For example, let $V = \{a, b, c\}$. Then since aab and cc are members of V^* by 2, $\{aab\}$ and $\{cc\}$ are regular languages. By 3, so is their union, $\{aab, cc\}$. By 4, so is their concatenation $\{aabcc\}$. Likewise, by 5 $\{aab\}^* \{cc\}^*$ are regular languages.

25.0.7. Pumping Lemma For instance, a non-regular language is, e.g., $L = \{a^n b^n \mid n \geqslant 0\}$. More generally, FSA cannot generate/recognize balanced open and closed parentheses.

You can prove that L is not a regular language by means of the **Pumping Lemma**.

Roughly note that with FSA you cannot record (**no memory**!) any arbitrary number of a's you have read, hence you cannot control that the number of a's and b's has to be the same. In other words, you cannot account for the fact that there exists a **relation of dependency** between a^n and b^n.

See Calvanese' s course for formal details.