## Contents

1. Introduction ................................................................................. 5
2. Rough Schedule ........................................................................ 6
3. Goals ....................................................................................... 7
4. Expected learning outcomes ..................................................... 8
5. Teaching Methods .................................................................... 9
6. Grading .................................................................................... 10
7. Your background ..................................................................... 11
8. Goals of Computational Linguistics ................................. 12
9. Why computational models of NL ...................................... 13
10. Challenges ............................................................................. 14
    10.1 Ambiguity: Phonology .................................................. 14
    10.2 Ambiguity: Morphology ................................................. 14
    10.3 Ambiguity: Syntax ........................................................ 15
    10.4 Ambiguity: Semantics .................................................... 16
    10.5 Ambiguity: Discourse ..................................................... 17
11. Challenges (Slide by F. Segond (Xerox)) ......................... 18
12. Challenges: (ditto) ................................................................. 19
1. Introduction

- **Time**: Mondays (10:30-12:30???), Wednesdays and Thursdays (13:00-15:00).

- **Office hours**: by appointment before or after classes.

- **Course Materials**: Slides, SLP text book (below), scientific papers

- **Text Book**

  1. D. Jurafsky and J. H. Martin *Speech and Language Processing*. (see nr. of chapters on the web).
1. **Introduction**

- **Time:** Mondays (10:30-12:30?), Wednesdays and Thursdays (13:00-15:00).

- **Office hours:** by appointment before or after classes.

- **Course Materials:** Slides, SLP text book (below), scientific papers

- **Text Book**

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

- **Url:** [http://www.disi.unitn.it/~bernardi/Courses/CompLing/18-19.html](http://www.disi.unitn.it/~bernardi/Courses/CompLing/18-19.html)
2. **Rough Schedule**

- 8 classes on Syntax (Sep-Oct): Formal Grammars of English, Syntactic Parsing, Statistical Parsing and Dependency Parsing (special guest: Barbara Plank!).


- 3 classes on Beyond sentences (Nov): Discourse Coherence, Dialogue, Question Answering/IQA (TBD) — Maybe something on evaluation measures.

- 5 classes on Multimodal Models (Dec): Language and Vision

Luca Ducceschi and I will coordinate our courses so that with him you have hands-on experience on the concepts you hear from me.
3. Goals

1. provide students with an overview of the field with focus on the syntax-semantics interface;

2. bring students to be aware on the one hand of several lexicalized formal grammars, on the other hand of computational semantics models and be able to combine some of them to capture the natural language syntax-semantics interface;

3. evaluate several applications with a special focus to Interactive Question Answering and Language and Vision Models;

4. make students acquainted with writing scientific reports.

All these objectives will help students understand how methods from computer science, mathematics and statistics are applied to the modelling of natural language and start being propositive for new ideas.
4. **Expected learning outcomes**

At the end of the course students will be able to:

1. illustrate the main challenges addressed in the field, which are its consolidated results and which are the current research questions;

2. master, at introductory level, the basic rules of some formal grammars and of formal and distributional semantics languages and their integration based on the principle of compositionality;

3. compare approaches on computational linguistics tasks, in particular within interactive question answering and language and vision integration;

4. apply interdisciplinary approaches to linguistics tasks and write a scientific report on their research in LaTeX.
5. Teaching Methods

We will have

1. frontal classes, pen-and-pencil exercises, discussions on papers lead by students (summary in LaTeX).

2. The latter will help students better grasp the basic rules of lexicalized formal grammar and formal and distributional semantics and their integration.

3. Students will be individually supervised on a project of their choice, to be selected on the base of their background and interest, in one of the topics discussed during the frontal classes. (suggestion: decide about this in November.)

4. Students will be supervised on the writing of a scientific report in LaTeX. During the course, each student has to hand-in at least the summary of one paper on syntax and one paper on semantics.
6. Grading

The final grade will be computed by the two grades below

1. **Written Exam (50%)**: exercises on Syntax, Semantics and their interface.
6. Grading

The final grade will be computed by the two grades below

1. **Written Exam (50%)**: exercises on Syntax, Semantics and their interface.

2. **Term paper (50%)**: You are to complete a project on topics of your choice upon agreement with me. The term paper has to be sent by mail to me by the day of the written exam. During the last classes, we will have project proposals presentations.

3. **Term paper expectation** In the project report, students will show they are able to compare approaches to computational linguistics tasks. The term paper must present an open problem in the CL field, review the relevant SoA and describe a proposal to address the problem or report about a project on it. It is meant to verify that students are able to read and understand technical works in computational linguistics, and to apply the relevant knowledge in a critical manner, showing they have learned how to reason in an interdisciplinary setting and write a scientific report in LaTeX.
7. Your background

Your background:

- Logic (PL?, FoL?)? (do you attend LSNL with RZ?),
- Formal Semantics, Distributional Semantics (Vector Space Semantics)?
- Programming skills? Python?
8. Goals of Computational Linguistics

- **Ultimate goal**: To build computer systems that perform as well at using natural language as humans do.
8. 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.
8. 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.
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 psycholinguist.
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 psycholinguist.

▶ **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.
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 psycholinguist.

- **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. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving ambiguity in one of the NL’s categories.
10. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving ambiguity in one of the NL’s categories.

An input is ambiguous if there are multiple alternative linguistic structures that can be built for it.
10. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving ambiguity in one of the NL’s categories.

An input is ambiguous if there are multiple alternative linguistic structures that can be built for it.

NL’s areas:
10. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving **ambiguity** in one of the NL’s categories.

An input is ambiguous if there are multiple alternative linguistic structures that can be built for it.

**NL’s areas:**

► Phonology – The study of linguistic sounds
10. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving *ambiguity* in one of the NL’s categories.

An input is ambiguous if there are multiple alternative linguistic structures that can be built for it.

**NL’s areas:**

- Phonology – The study of linguistic sounds
- Morphology – The study of the meaningful components of words
10. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving **ambiguity** in one of the NL’s categories.

An input is ambiguous if there are multiple alternative linguistic structures that can be built for it.

**NL’s areas:**

- **Phonology** – The study of linguistic sounds
- **Morphology** – The study of the meaningful components of words
- **Syntax** – The study of the structural relationships between words
10. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving *ambiguity* in one of the NL’s categories.

An input is ambiguous if there are multiple alternative linguistic structures that can be built for it.

**NL’s areas:**

- **Phonology** – The study of linguistic sounds
- **Morphology** – The study of the meaningful components of words
- **Syntax** – The study of the structural relationships between words
- **Semantics** – The study of meaning
10. Challenges

In short, most or all tasks of speech and text processing can be viewed as resolving **ambiguity** in one of the NL’s categories.

An input is ambiguous if there are multiple alternative linguistic structures that can be built for it.

NL’s areas:

- **Phonology** – The study of linguistic sounds
- **Morphology** – The study of the meaningful components of words
- **Syntax** – The study of the structural relationships between words
- **Semantics** – The study of meaning
- **Discourse** – The study of linguistic units larger than a single utterance
10.1. Ambiguity: Phonology

Phonology: It concerns how words are related to the sounds that realize them. It’s important for speech-based systems.
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”
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”
10.1. 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)
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
10.1. 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
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)]
10.3. Ambiguity: Syntax

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

1. I saw the man with the telescope

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

   ▶ $[I[[\text{saw}]_v[[\text{the man}]_{np}[\text{with the telescope}]_{pp}]_{np}]_{vp}]_s$  
   [(the man has the telescope)]
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.
10.3. Ambiguity: Syntax

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

1. I saw the man with the telescope

   - $[I[[saw]_v[the\ \text{man}]_{np}[with\ \text{the\ telescope}]_{pp}\_vp]_s$  
     
     $[(I\ \text{have\ the\ telescope})]$ 
   
   - $[I[[saw]_v[[the\ \text{man}]_{np}[with\ \text{the\ telescope}]_{pp}\_np]\_vp]_s$  
     
     $[(the\ \text{man\ has\ the\ telescope})]$ 

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.
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.
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 ...?
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.
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 Prodomo Ltd.

(a joint venture!\textsubscript{i})
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.
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** will be called **Prodome Ltd.**
   (a joint venture!)

3. **Merck & Co.** formed a joint venture with **Ache Group**, of Brazil. **It** will own 50% of the new company to be called **Prodome Ltd.**
   (Merck & Co.)!
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*\textsubscript{i} will be called Prodome Ltd. 

(a joint venture!\textsubscript{i})

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

(Merck & Co.\textsubscript{i}!)

4. Merck & Co. formed a joint venture with Ache Group, of Brazil. *It*\textsubscript{i} had previously teamed up with Merck in two unsuccessful pharmaceutical ventures.
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 will be called Prodome Ltd. (a joint venture!)

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

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

---

Contents  First  Last  Prev  Next  ◄
11. Challenges (Slide by F. Segond (Xerox))

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?

The difficulty

Mary takes a glass of wine
Mary takes a gym course
Mary takes a husband

<table>
<thead>
<tr>
<th>actor</th>
<th>experiencer</th>
</tr>
</thead>
</table>

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

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".
12. Challenges: (ditto)

KIDS MAKE NUTRITIOUS SNACKS

- You had kids for appetizers?
- No! Kids are good at cooking snacks
- Really? They transformed into powder??
- No! they use powdered milk

MILK DRINKERS ARE TURNING TO POWDER

- Oh my God! Did she go to the hospital?
- No! The boat was repaired
- Really? They transformed into powder??

QUEEN MARY HAVING BOTTOM SCRAPED
13. NLP Systems

1. Tokenization
13. NLP Systems

1. Tokenization

2. PoS tagging
13. NLP Systems

1. Tokenization
2. PoS tagging
3. Morphological analysis
13. NLP Systems

1. Tokenization
2. PoS tagging
3. Morphological analysis
4. Shallow parsing
13. NLP Systems

1. Tokenization
2. PoS tagging
3. Morphological analysis
4. Shallow parsing
5. Deep parsing
13. NLP Systems

1. Tokenization
2. PoS tagging
3. Morphological analysis
4. Shallow parsing
5. Deep parsing
6. Semantic representation (of sentences)
13. NLP Systems

1. Tokenization
2. PoS tagging
3. Morphological analysis
4. Shallow parsing
5. Deep parsing
6. Semantic representation (of sentences)
7. Discourse representation
13. NLP Systems

1. Tokenization
2. PoS tagging
3. Morphological analysis
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*).
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”.
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”.
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.
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?
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?
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.
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).
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.
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.
16. Morphology

Morphology is the study of how words are built up from smaller meaning-bearing units, morphemes. It concerns the inner structure of words.
16. 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
16. 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.
16.1. Morphemes

Morphemes are divided into:

1. **stems**: they are the main morpheme of the word, supplying the main meaning.
16.1. 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:
16.1. 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)
16.1. 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)
16.1. 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)
16.1. 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))
16.1. 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).
16.2. Ways of forming new words

There are two basic ways used to form new words:
16.2. 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,
16.2. **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”.
16.2. 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.
16.2. 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.
16.2. 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.
16.2. 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”.
17. 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.
17. 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.

17.1. Modules

To build a morphological recognizer/generator, we’ll need at least the following:
17. Computational Morphology

We want to build a system able to provide the stem and the affixes given a word as input (e.g. cats → \{\text{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.

17.1. 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).
17. 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.

17.1. 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.
17. 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.

17.1. 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”.

---

Contents   First   Last   Prev   Next
17.2. 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.
17.2. 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).
18. **In short: Formal Language & FSA**

We will briefly look at how FSA are used to recognize/generate natural language morphology.

First we go through some background notions, namely **Formal Languages** and FSA.

- A formal language is a set of strings. E.g. \{a, b, c\}, \{the, a, student, students\}.
18. **In short: Formal Language & FSA**

We will briefly look at how FSA are used to recognize/generate natural language morphology.

First we go through some background notions, namely Formal Languages and 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.
18. **In short: Formal Language & FSA**

We will briefly look at how FSA are used to recognize/generate natural language morphology.

First we go through some background notions, namely **Formal Languages** and FSA.

- A formal language is a set of strings. E.g. \( \{a, b, c\} \), \( \{\text{the, a, student, students}\} \).
- 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.
18. In short: Formal Language & FSA

We will briefly look at how FSA are used to recognize/generate natural language morphology.

First we go through some background notions, namely Formal Languages and 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.
18. In short: Formal Language & FSA

We will briefly look at how FSA are used to recognize/generate natural language morphology.

First we go through some background notions, namely Formal Languages and 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 Languages.
18.1. Finite State Automata

A finite state **generator** is a simple computing machine that **outputs** a sequence of symbols.
18.1. 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.
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?
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.
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.

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.
18.1.1. 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:

![Diagram](image)

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.
18.1.2. 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 initial 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.
18.1.2. 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
18.1.2. 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 initial 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.
18.1.3. Recognizer: an example  So, this machine recognizes a laughter.
18.1.3. 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.
18.1.4. Finite State Automata  Try to think of what language is recognized or generated by the FSA below.
18.1.5. Finite State Automata with jumps
18.1.5. 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 $\epsilon$ 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.
18.1.6. 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.
18.1.6. 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.
18.1.6. 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 initial and final states (it must have at least one initial and one final state, though).
18.2. Regular Languages

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 $V$, 
18.2. Regular Languages

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. $\emptyset$ is a regular language
18.2. Regular Languages

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.
18.2. Regular Languages

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

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

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

18.2. Regular Languages

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 $AB$. 
18.2. Regular Languages

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 $A^*$.
18.2. Regular Languages

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.
18.2. Regular Languages

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.
18.2.1. **Pumping Lemma** For instance, a non-regular language is, e.g., $L = \{a^n b^n \mid n \geq 0\}$. More generally, FSA cannot generate/recognize balanced open and closed parentheses.
18.2.1. Pumping Lemma  For instance, a non-regular language is, e.g., \( L = \{a^n b^n \mid n \geq 0\} \). 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**.
18.2.1. Pumping Lemma  For instance, a non-regular language is, e.g., $L = \{a^n b^n \mid n \geq 0 \}$. 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 $b^n$. 
19. 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.
19. 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.
19. 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.

    Stem Language: \{work, talk, walk\}. 
19. 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.

  Stem Language: \{work, talk, walk\}.
  Suffix Language: \{ε, −ed, −ing, −s\}.
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.

  Stem Language: \{work, talk, walk\}.
  Suffix Language: \{ε, −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\}
19.1. FSA for English Inflectional Morphology

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

<table>
<thead>
<tr>
<th>reg-stem</th>
<th>plural</th>
<th>pl-irreg-stem</th>
<th>sing-irreg-stem</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>-s</td>
<td>geese</td>
<td>goose</td>
</tr>
<tr>
<td>cat</td>
<td></td>
<td>sheep</td>
<td>sheep</td>
</tr>
<tr>
<td>dog</td>
<td></td>
<td>mice</td>
<td>mouse</td>
</tr>
</tbody>
</table>
19.1. FSA for English Inflectional Morphology

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

<table>
<thead>
<tr>
<th>reg-stem</th>
<th>plural</th>
<th>pl-irreg-stem</th>
<th>sing-irreg-stem</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>-s</td>
<td>geese</td>
<td>goose</td>
</tr>
<tr>
<td>cat</td>
<td></td>
<td>sheep</td>
<td>sheep</td>
</tr>
<tr>
<td>dog</td>
<td></td>
<td>mice</td>
<td>mouse</td>
</tr>
</tbody>
</table>

![Diagram of FSA for English nominal inflection](image-url)
Let's build an FSA that recognizes English adjectives. Our lexicon is:

<table>
<thead>
<tr>
<th>adj-root1</th>
<th>adj-root2</th>
<th>Suffix-1-2</th>
<th>Suffix-1</th>
<th>Affix-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>big</td>
<td>-er</td>
<td>-ly</td>
<td>un-</td>
</tr>
<tr>
<td>happy</td>
<td>cool</td>
<td>-est</td>
<td></td>
<td></td>
</tr>
<tr>
<td>real</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
20. Recognizers vs. Parsers
20. 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
20. 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).
20. 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.
20.1. 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 \{\text{cat N PL}\}.
21. 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:
21. 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
21. 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.
22. Conclusion

▶ Today we have introduced CL
▶ On Thursday, we will look at Syntax (Context Free Grammars, CFG)
▶ Next week, we will do exercises on CFG and introduce LaTeX (Who knows it already?)
▶ Schedule: would it be possible to have classes on Fridays instead of Mondays (e.g. 13:30-16:30)? – 12:00-12:30 Brown Bag meetings)
▶ Clic: We will have a clic meeting on the 10th, with presentations of current research projects.