Summary

Grading:

- 2 individual tasks (10% both)
- Group assignment (not graded) - programing assignment
- Final exam (80%) . 2.30h - multiplechoice

Chapter 1. **Introduction**

27/08/18

Trying to make the computer understand language. Not the programming languages but natural languages. We want the computer to understand natural languages. The goal of this new field is to get computers to perform useful tasks involving human languages, tasks like enabling human-machine communication, improving human-human communication, or simply doing useful processing of text or speech.

Components:

- **Morphology**: analyzes the structure of the words itself.
- **Syntax**: sentence structure. The order, and the rules on how to put words together to form a proper sentence. What combinations are valid.
- **Semantics:** meaning.
  - Lexical semantics: word meaning
  - Compositional semantics: meaning of larger structures
- **Pragmatics:** not the actual meaning but what is intended by the sentence. “It is cold” - I want you to close the window. Intended meaning of the sentence
- **Discourse**: knowledge of units larger than a sentence: multiple sentences in a higher level.

Computer does not understand anything about the language. Just numbers. How can we get the computer to understand what a word is?

Models and algorithms

Models. Figure out what the rules are from the linguistic perspective. Math models that allow to describe what words are. Models:

- State machines (can be augmented with prob models)
- Rule systems (can be augmented with prob models)
- Logic (can be augmented with prob models)
- Probabilistic models
- Vector-space models

Probabilistic models are crucial for capturing every kind of linguistic knowledge. Each of the other models (state machines, formal rule systems, and logic) can be augmented with probabilities. For example, the State Machines can be augmented with probabilities to become **weighed automaton** or **Markov model**.

We will spend some time on **hidden Markov models** or **HMMs,** which are used everywhere in the field, in part-speech tagging, speech recognition, dialogue understanding, text-to-speech, and machine translation.

The key advantage of probabilistic models is their ability to solve ambiguity problems. Almost any speech and language processing can be recast as "given N choices for some ambiguous input, choose the most probable one".

Imagine we find the right model (allows to describe something) and we need to figure out how to use the model to make it work — then we have **algorithms** (they extract info from the text). Therefore → Models describe / algorithms analyze

Algorithms:

- Sate space search (dynamic programming)
- Machine Learning [ex. Classifiers (decision trees, support vector machine, Gaussian mixture models, logistic regression) and sequence models (hidden Markov models)].
  Classifiers can assign a single object to a single class. Sequence models can do the same but classify a lot of objects at the same time, putting them into different classes.

When are we doing a good job? When does a computer understand language?
Alan Turing. (1912-1954) Do we understand what understanding is?

Turing Test (1950): you type in the computer or screen communicating to someone in the other room and the idea is that the computer is so good that you can't tell the difference. If you can fool them, then the computer is good enough. We are not close.

Difficulties - Ambiguity

we still don't know all the rules of language & Language are ambiguous:

- Lexical Ambiguity: party // context helps

- Part of Speech (POS) ambiguity: Ex. Times flies like an arrow - Time (n) flies (v) like (adj) arrow (n). But like can also be a V and flies can also be a N:
  Time flies like an arrow
  Fruit flies like a banana
  Here context is not enough. Here we need a step further.

- Syntactic scope: PP attachment
  Give me the list of products in 2008 - the list of what and give it when???
  The man sees the women with the binoculars
  We need **more external** knowledge here.

- Syntactic scope: conjunctions
  This field should contain last name and first name or initials. — Last name and initials ? Initials only? — We need to **re write** the whole sentence.

- Syntactic scope: quantifiers
  All farmers beat a donkey - How many donkeys are there?

- Pragmatic
  Shoes must be worn - Always
  Dogs must be carried – Only when you have a dog

We need to have some common sense & world knowledge.

Foundational insights – History – LLEGIR

1940s and 50s. War time. It was mostly theoretical.

In 1970-1983 symbolic school is spited in 3 other schools: logic / NLP understanding. Discourse modeling.

1983 - 1993

1994 - everything became probabilistic

2000-now - Machine Leaning - DL (going back to the neural nets that we had in the 60's, but now computers are fast enough to actually use them).

___

Foundation insights

Automaton

- formal languages
- finite-state automata (next lecture)
- regular expressions
- context-free grammars

Probabilistic/information-theoretic models

- Claude Shannon (1916- 2001)
- Noisy channel/entropy
- Information theory

Two camps (1957 – 1970)

- **Symbolic:** Based on formal grammars. Parsing algorithms
- **Stochastic:** Probabilistic models (Bayesian models). Neural nets

Four paradigms (1970 – 1983)

- **Stochastic**: Hidden Markov model (HMM)
- **Logic-based:** Feature structure unification
- **Natural Language Understanding:** SHRDLU
- **Disclosure modeling:** Structure in discourse

Empiricism and finite-state model redux (1983 – 1993)

- **Finite-state models**
  - Morphology
  - Phonology
  - Syntax
- **Probabilistic models**
  - Speech recognition
  - POS tagging
  - Parsing
  - Semantics
  - Evaluation

Field comes together (1994 – 1999)

Probabilistic aspects in many algorithms/models
Applications arise
Increase in computational speed and memory size
Need for large scale NLP (ex. IR on the web)

Rise of ML (2000 – now)

Large amounts of data available (speech, text)
Statistical ML
High performance computing facilities
Unsupervised statistical learning approaches
Deep Learning, Big Data

Chapter 2. **Regular expressions and automata**

30/08/18

Index

Regular Expressions (RE)

Corpus: A collection of text
String: A sequence of symbols/characters (alphanumeric). Space and Newline are also characters.

Regular Expressions is a notion to characterize (sub)strings. They are used to search for pattern occurrences in a corpus. Example: /(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/!/	"You've left the burglar behind again!", said Nori

Regular expressions are case sensitive: lowercase /s/ is not the same as uppercase **/S/.** We can solve this problem with the use of square braces [and]. The string inside these braces specifies a **disjunction** of characters to match.

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[1234567890]	Any digit	"Plenty of 7 to 5"

Regular expression/Regex/RE

/a/ Matches the first occurrence of the letter 'a'
/abc/ Matches 'abc' (concatenation)

- RE match the first exact occurrence in a string
- RE are case-sensitive: lower-case ≠ upper-case

Example

/c/ abcdef
/c/ abcdefc
/abc/ ab c abc abc
/ab c/ ab c abc abc
/abc/ ABC Abc aBc abc abc

Regular expressions that specify any digit or any capital letter, can get inconvenient. In these cases, brackets can be used with the dash (-) to specify any one character in a **range.**

RE	Match	Example Patterns Matched
/[A-Z]	An uppercase letter	"We should call it 'Drenched Blossoms'"
/[0-9]/	A single digit	"Chapter 1: Down the Rabbit Hole"

The square braces can also be used to specify what a single **character cannot be**, by use of the caret. For example, the pattern **/[^a]/** matches any single character except *a*. If the caret occurs anywhere else (not after the open square brace), it usually stands for a caret.

RE	Match	Example Patterns Matched
[^Ss]	Neither S nor s	"I have no exquisite reason for 't'"
[e^]	Either 'e' or '^'	"Look up ^ now"

To specify, for example, both woodchuck and woodchucks, we use the question mark **/?/:** the preceding character of nothing. Question mark ? marks optionally of the previous expression.

RE	Match	Example Patterns Matched
woodchucks?	woodchuck or woodchucks	"woodchuck"
colou?r	color or colour	"colour"

For repetition, we use the **Kleene ,** which means "zero or more occurrences of the immediately previous character or regular expressions". So, **/a*/** matches with a, aaaa, but also with Off Minor, since the string has zero a's. So, the regular expression for matching one or more a is **/aa*/**

/[ab]*/ means "zero or more a's or b's. This will match strings like aaaa, ababab or bbbb

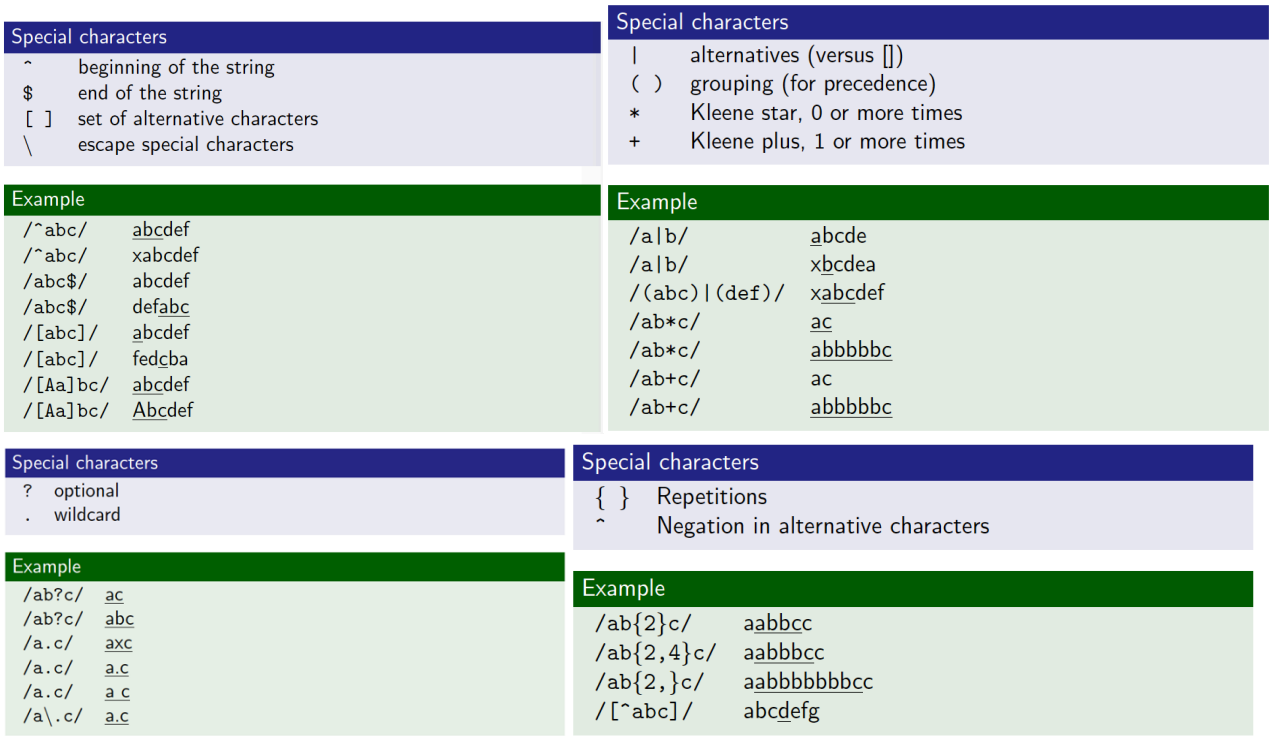
Kleene + which means “one or more of the previous characters”, is also used for repetition. – LLEGIR p 54

One very important special character is the **period** (/./), a wildcard expression that matches any character.

__M ‘He quedat a la pag 55

Anchors are special characters that anchor regular expressions to particular places in a string. The most common ones are the caret ^ (the start of a line) and the dollar sign $ (the end of a line). So, the pattern /^The/ matches the word *The*, only at the start of a line. The pattern /^The dog\.$/ matches a line that contains only the phrase *The dog.*
There are two more anchors: \b matches a word boundary, while \B matches a non-boundary. Thus, /\bthe\b/ matches the word “the”, but not the word “other”.

A **disjunction** or **pipe** is used to specify for example the string cat or the string dog. This will look like this: /cat|dog/.



Shortcuts	
`[ - ]`	Ranges
`\n`	Newline
`\t`	Tab
`\b`	Word boundary
`\B`	Word non-boundary
`\d`	Digit (`/[0-9]/`)
`\D`	Non-digit (`/[^0-9]/`)
`\w`	Alphanumeric or underscore (`/[a-zA-Z0-9_]/`)
`\W`	`/[^a-zA-Z0-9_]/`
`\s`	Whitespace (including newline)
`\S`	Non-whitespace (including newline)

..

Exercise
Write a regular expression for:

- **the set of all lower case alphabetic strings ending in a "b"**

*. period maches any carater ^indicates the start of the string $ this is the end of the text * 0 or more*

/^./ —> this would select everything. not what we want
/^[a-z]*b$/ —> $ couse it should end with the b. We still need to indicate the beginning with a ^ before the brakets. We need to indicate that where it starts couse we dont want any numbers or capital laters in our word.

- **telephone number.** We want to create something that accepts all t numbers and nothing else:

country code - +?
area code
dash something
amount of digits
number
short numbers (emergency)

Let's start simple. Just the number (7 in Tilburg)

/^\d {6,7}$/** (6,7) means that it can be 6 or 7 numbers or **/^\d \d \d \d \d \?d$/

ex: 013-4661234 or 0134-123456 or without slash
/^\d{3}[-?]\d{7}| \d{4}[-?]\d{6}$/

Regular expressions describes a particular language. A formal language.

The idea is to describe natural languages with formal languages.
A language is a set of strings

We can have a regular language that just accept one character. (not really useful) I can start the combining languages.

(L1)A language in this sense is a set of strings. We define languages.

exercise: I have a language that just have 3 strings: a, aa, ab

L_1={a}
L_2= {b}
L_1L_1={aa}
L_1L_2={ab}

result —> {a, aa, ab} = $L_1 U L_1L_1 U L_1L_2$

q are states.
sigma is all the symbols
F is the final state
(la taula) recognize step by step and if you recognize it you can go to the next state until you reach the F

Deterministic: you always know what to do. If you find it you keep going.
Non-deterministic: there is a state where you don't know exactly where to go. There is a choice at some point. epselon transition

Chapter 4. **N-Grams**

3/09/18

Word prediction with a probabilistic model called **N-grams model**. It predicts the next word from the previous N – 1 word. N-gram is a **N-token** sequence of words. **Bigram / Trigram.**

The guessing given knowledge of counts on n-grams can be done by:

- Compute conditional probability of candidate words
- Compute probability of an entire sequence

The order of the words matters. These kind of statistical models of word sequences are called **Language Models (LMs).**

$P(w_n | w_1, w_2, \ldots, w_{n-1})$

Applications:

- Automatic speech character recognition
- Spelling correction
- Machine Translation

They picnicked by the pool, then lay back on the grass and looked at the stairs. →
18 tokens (counting punctuation), **16 types**.

4.1 Word Counting in Corpora

Corpus (plural: **corpora**)

Brown University (1992) and Switchboard (telephone - spoken conversations). Google n-gram corpus (crawl of 1,024,908,267,229 English **tokens** // 13,588,391 wordform **types**)

Example sentence

I do uh main- mainly business data processing.

Two types of **diffluences**: **Filler** (uh) and **Fragment** (main-). Should we keep the diffluences? Depends on the application.

Should we count “cat” / “cats” as the same? “geese” / “goose”? → depends on the application
Same lemma but two different **wordforms**.
Lemma: a set of lexical forms having the same stem, major part of speech, and rough word sense

4.2 Simple (unsmoothed) N-Grams

Conditional probability:

Use definition of conditional probabilities
Look for counts
$\frac{c(w_1,w_2,\ldots,w_{n-1},w_n)}{c(w_1,w_2,\ldots,w_{n-1})}$
Get counts from large corpus

Problem: Longer sentences lead to low counts.
What happens when the count is 0? Or both are 0?
Solution:

- Use **chain rule of probability:** The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. We can estimate the joint probability of an entire sequence of words by multiplying a number of conditional probabilities. → It is not useful. We don't know any way to compute the exact probability of a word given a sequence.
- Particularly useful independence assumption.
  - **Markov assumption**: probability of words depends only on the N previous words (N-grams)

    Bigrams $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$ *General:* $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$
  - **MLE** (Maximum Likelihood Estimation) to estimate probabilities.

    Bigrams: $P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$ *General:* in the book / p. 123

Formula Chain Rule:

Chain rule

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1}) \\ &= \Pi_{k=1}^n P(w_k|w_1^{k-1}) \end{aligned}$$

Example

$$\begin{aligned} P(\text{its water was so transparent}) &= P(\text{its}) \\ &\times P(\text{water}|\text{its}) \\ &\times P(\text{was}|\text{its water}) \\ &\times P(\text{so}|\text{its water was}) \\ &\times P(\text{transparent}|\text{its water was so}) \end{aligned}$$

Example of a MLE (**relative frequency**):

Example

⟨s⟩ I am Sam ⟨/s⟩
⟨s⟩ Sam I am ⟨/s⟩
⟨s⟩ I do not like green eggs and ham ⟨/s⟩

$P(\text{I}|\langle\text{s}\rangle) = \frac{2}{3} = .67$ $P(\text{Sam}|\langle\text{s}\rangle) = \frac{1}{3} = .33$
$P(\text{am}|\text{I}) = \frac{2}{3} = .67$ $P(\langle/\text{s}\rangle|\text{Sam}) = \frac{1}{2} = .50$
$P(\text{Sam}|\text{am}) = \frac{1}{2} = .50$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{\text{count}(w_{n-N+1}^{n-1}, w_n)}{\text{count}(w_{n-N+1}^{n-1})}$$

These probabilities can tell us a lot about human behavior:

Kinds of knowledge

n-gram probabilities capture range of interesting facts

- World knowledge
  - $P(\text{english}|\text{want}) = .0011$
  - $P(\text{chinese}|\text{want}) = .0065$
- Syntax
  - $P(\text{to}|\text{want}) = .66$
  - $P(\text{eat}|\text{to}) = .28$
  - $P(\text{food}|\text{to}) = 0$
  - $P(\text{want}|\text{spend}) = 0$
- Discourse
  - $P(\text{i}|\langle\text{s}\rangle) = .25$

Interesting matrix and a full **FULL example** of the probably of ***I want English food → pg 125***

Normally, **trigram models** are used when we have sufficient training data.

We can also **generate sequences**, using the Shannon:

Generation (Shannon)

1. Sample random bigram ($\langle$s$\rangle$, w) according to its probability
2. Now sample a random bigram (w, x) according to its probability
3. Where the prefix w matches the suffix of the first
4. And so on until we randomly choose a (y, $\langle$/s$\rangle$)
5. Then string the words together

Here the result of applying the technique of Shannon (1951): - pg 127

Bigram → What means, sir. I confess she? then all sorts, he is trim, captain. Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.

Trigram → Sweet prince, Falstaff shall die. Harry of Monmouth's grave. This shall forbid it should be branded, if renown made it empty. Indeed the duke; and had a very good friend. Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram → King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; Will you not tell me who I am? It cannot be but so. Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare as a corpus

- N=884,647 tokens, V=29,066 types
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams...
- So, 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Biggest problem in language modeling; we'll come back to it
- Quadrigrams are worse: What's coming out looks like Shakespeare because it *is* Shakespeare

Number of possible bigrams = V^2
Number of possible quadrigrams = V^4

4.3 Training and Training Sets

Training Set | **Test Set** → used to evaluate different N-gram models.
Check "held-out" and "development test set" (sevset) – pag 126

Perplexity: Evaluation Metric that tells how well a given statistical model matches a test set. It is based on computing the probability of each sentence in the test set: intuitively, whichever model assigns a higher probability to the test set.

Training Data Sets with Shakespeare vs Wall Street Journal → stark difference
Be sure to use training corpus that look alike to test corpus.

Closed Vocabulary: The test can only contain words from the lexicon. It assumes that there are no unknown words. → this is a simplification. The number of unseen words grow constantly. These are called **OOV (Out Of Vocabulary words).** → the percentage of OOV words that appear in the test set is called **OOV rate.**

Open Vocabulary system: we module these potential unknown words in the test set by adding a pseudo-word token called <UNK>:

- Training of <UNK> probabilities:
  - Create a fixed lexicon L of size V
  - At text normalization phase:
    - Any training word not in L changed to <UNK>
  - Now we train its probabilities like a normal word
- Decoding time:
  - If text input:
    - Use <UNK> probabilities for any word not in training

**The closed vocabulary task specifies the vocabulary for the test set in advance. This can significantly reduce perplexity. In general, perplexity between 2 models is only comparable if they use the same vocabulary.*

4.4 Evaluation N-Grams: Perplexity (PP) (intrinsic evaluation)

The best way to evaluate the performance of a LM (language model) is to embed it in an application and measure the total performance of the application. Such end-to-end [often very expensive] evaluation is called **extrinsic evaluation** or **in vivo**. → Compares the performance of system using different language models. Expensive and very time-consuming.

Another quick variable is **intrinsic evaluation (in-vitro)**. It measures the quality of the model independent of any application. → **Perplexity (PP)** is the most common one for N-gram models.

While and improvement in intrinsic evaluation does not guarantee an extrinsic improvement, they often correlate. (1st) Perplexity for a quick check and after, apply an (2nd) end-to-end evaluation for a final confirmation.

How does PP work? It is the probability of test set.
The best model is the one that can predict better the test set. Tighter fit.
The PP of a LM on a test set is a function of the probability that the language model assigns to that test. We want to minimize perplexity. The more information the N-gram gives us about the word sequence, the lower the perplexity. **Trigram model will always have a lower perplexity than a bigram model.**

- Perplexity is probability of test set
  - assigned by the language model
  - normalized by the number of words

$$PP(W) = P(w_1 w_2 \dots w_n)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_n)}}$$

- Chain rule:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

- Bigrams:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

- Minimizing perplexity is same as maximizing probability

*The closed vocabulary task specifies the vocabulary for the test set in advance. This can significantly reduce perplexity. In general, perplexity between 2 models is only comparable if they use the same vocabulary.

4.5 Smoothing

MLE has a problem:

- Zero counts
- When non-zero but still small it produces poor estimations.

- The perplexity metric requires that we compute the probability of each test sentence. But if the test sentence has an N-gram that never appeared in the training set, the MLE of the probability for the N-gram, and hence for the whole test set, will be 0. → we cannot evaluate the model if we do not assign some non-0 values to the N-grams that never appear in the training set.

Laplace Smoothing
Take the matrix of bigrams counts before normalizing it into probabilities and 1 to all the counts.

Sharp changes in counts and probabilities occurs because too much probability mass is moved to all the zeros. For this, and other reasons, we'll need better smoothing methods for N-grams like:

- Good-Turing
- Kneser-Ney
- Witten-Bell

Laplace still used for pilot studies and when there are not many zeros.

Laplace or add-one smoothing

- Just add one to all the counts
- MLE estimate: $P(w_i) = \frac{c_i}{N}$
- Laplace estimate: $P(w_i) = \frac{c_i+1}{N+V}$
- Reconstructed counts: $c_i^* = (c_i + 1)\frac{N}{N+V}$
- All zero counts become one (very low probability)

4.6 Interpolation

If we estimate P (z | x,y) but counts (xyz) is zero, we use info from P (z|y) or even P(z).
There are 2 ways of using N-grams "hieracy": **backoff** and **interpolation**.

Backoff vs Interpolation:

- In **backoff**, we use trigram if available, otherwise bigram, otherwise unigram
  If we have non-zero trigram counts, we rely solely on the trigram counts. We only "back off" to a lower-order N-gram if we have zero evidence for a higher-order N-grams.

- In **Interpolation** we combine result of all three
  We always mix the probability estimates from all the N-gram estimators, that is, we do a weighted interpolation of trigrams, bigrams and unigrams counts.

Suma de Lambda is 1.

In a more sophisticated way, each lambda is conditional on context. Adapt the weights according to the context. – p 138

- Simple interpolation

$$\begin{aligned} P'(w_n|w_{n-1}w_{n-2}) &= \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ &+ \lambda_2 P(w_n|w_{n-1}) \\ &+ \lambda_3 P(w_n) \\ \Sigma_i \lambda_i &= 1 \end{aligned}$$

- Lambda conditional on context:

$$\begin{aligned} P'(w_n|w_{n-1}w_{n-2}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-1}w_{n-2}) \\ &+ \lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &+ \lambda_3(w_{n-2}^{n-1}) P(w_n) \end{aligned}$$

How is lambda set? Lambda are learned from the **held-out** corpus [section 4.3]. A held-out corpus is an additional training corpus that we use, not to set the N-gram counts but to set other parameters. We chose the lambda values that maximize the probabilities of the held-out corpus. We fix n-gram probabilities and search for lambda values that gives us the higher likelihood in the simple interpolation function.

4.7 Backoff

Backoff is a more complicated, but better algorithm to use. The one that's used in the book is Katz backoff algorithm that uses Good-Turing discounting. In this model, if the N-gram we need has zero counts, we approximate it by backing off to the N-1 gram. We keep backing off until we reach some counts. In simple words it can be described as: Use trigram if available otherwise bigram, otherwise unigram.

This equation shows that the Katz backoff probability for an N-gram just relies on the (discounted) probability P* if we've seen this N-gram before (i.e. if we have non-zero counts). Otherwise, we back off to the Katz probability for (N-1)-gram. The following equation gives an example.

Because on average the (discounted) c* will be less than c, this probability P* will be slightly less than the MLE estimate.

4.8 Practical Issues: Toolkits and Data Formats

N-gram language models are represented in log format, to **avoid underflow** and to **speed up the computation.** The more probabilities we multiply together the smaller the product will be. Using log probabilities will prevent that the product will become to small (underflow).

$P1 \times P2 \times P3 \times P4$ = e^($logP1 + logP2 + logP3 + logP4$)
Backoff N-gram LM are usually stored in a **ARPA format** – p142

Practical issues

- All computations in log space
  - Avoid underflow
  - Faster

$$p_1 \times p_2 \times p_3 = e^{\log p_1 + p_2 + p_3}$$

For building language model, we use **toolkits**. There are two commonly used toolkits publicly available. Those are the **SRILM toolkit** and the **Cambridge-CMU toolkit**.

(Foto de exemples de probability)

We are doing almost the same as the last chapter thing but from another approach.
There are so many words and combinations of those that we can't use the same thing as last chapter. So, we use probability. How **likely is it that this particular sentence is part of my language**?

What are we counting? **Tokens**, which are words but also punctuation. Any symbol in your language. You can make choices there depending on what kind of apps you want to build there. What tokens are value to count?

Unique tokens are called **types**: the ones that do not repeat among the sentence
Cat / cats - depending on the app you want to count them as just one (it is called **limitezation**)

P (probability of a particular word will be next | all the words before the word we are interested in)
The c in the formula stands for "counts"

If you search for long patters or sequences the counts go down.

If a sentence is not in a corpus maybe we wrongly assume that it is not likely in our language, but the problem is that our corpus is not big enough.

The "**chain rule**" will give me the probability of the whole sentence. (Aqui els numeros a dalt no son quadrats, sino com la sequencia: del numero d abaix al numero d adalt)
.. but if I want to try these 0 counts, just by doing this chain rules does not help. — (**mark of assumption**: the way to solve this is by assuming that we do not need the whole context before the word, the probability of a certain word after a long sentence is aporx the same as the prob of a word after just the last word.) **You can't do a mark of assumption without doing the chain rule before.**

In the formula of P — the numerator is the context + the word i am interested in / the denominator is the context.

Perplexity PP : avoid the overfitted models. Allows me to compare prob of long sentences and short sentences. It normalizes the prob because of course longer sentences have less prob as every word added is multiplied by 0.smthing
Low perplexity is better than high perplexity.

Smoothing: remove some of the probabilities in the words that already are in our language and putting some to these new words.

There are diff ways of dealing with the 0 probabilities:

- mark of assumption
- Smoothing
- Backoff

Chapter 5. **Part-of-speech (POS) Tagging**

06/09/2018

Part-of-speech (or **tagsets**). The significance of POS for languages is the large amount of information they give about a word and its neighbors.

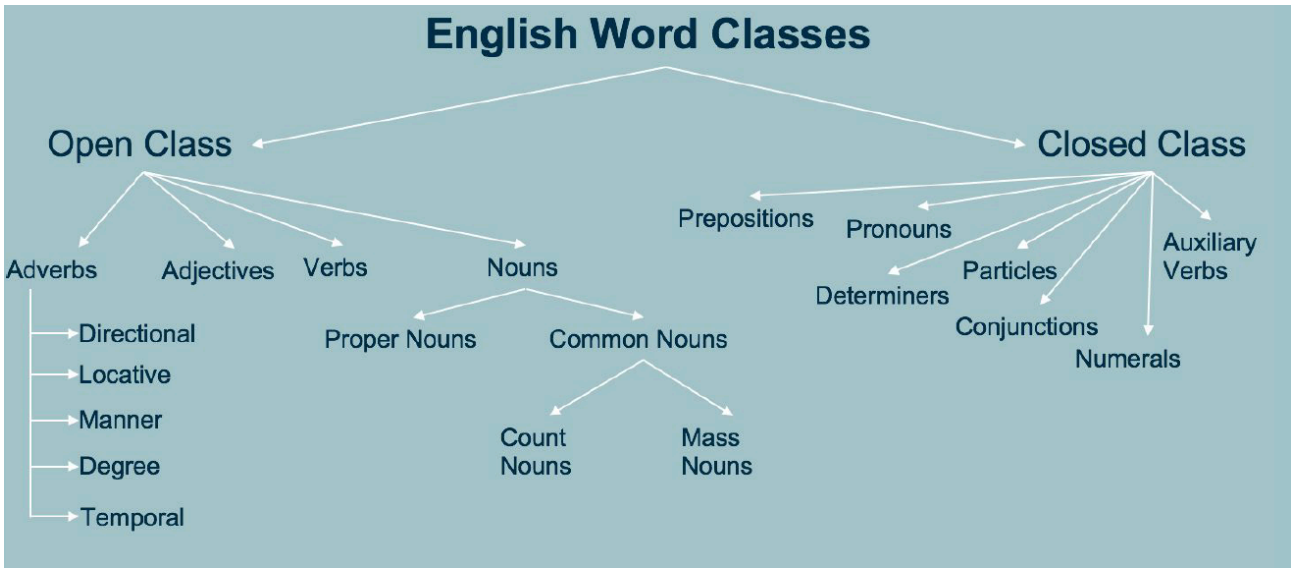
Pronunciation
CONtent (noun) vs. conTENT (adjective)
OBject (noun) vs. obJECT (verb)

Automatic assignment of POS plays a role in parsing, in **word sense disambiguation** algorithms, and in shallow **parsing** of text to quickly find names, times, dates, or other **named-entities recognition**, **information retrieval.**

This chapter focuses on computational methods for assigning parts-of-speech to words (**part-of-speech tagging**). Many algorithms have been applied to this problem, including hand-written rules (**rule-based tagging**), probabilistic methods (**HMM tagging** and **maximum entropy tagging**), as well as other methods such as **transformation-based tagging** and **memory-based tagging**.
We will introduce 3 of these algorithms.

- **Rule-based tagging**
- **HMM tagging** (Hidden Markov Models)
- **Transformation-based tagging**

5.1 (Mostly) English Word Classes



POS can be divided into two broad super-categories:

Open-Class POS:
Nouns: House, membership, Colorado, dream...
Verb: run, wait, see, speak, ...
Adjectives: red, good, beautiful
Adverbs: Unfortunately, John, extremely, slowly

Closed-Class POS: (relatively fixed membership) – they tend to be function words (short, occur frequently, and often have structuring uses in grammar)
Prepositions: *"the cage test" In the cage, on the cage, under the cage, over... (all the words that fit before the cage are prepositions*)
Determiners: a, an, the, many...
Pronouns: she, who, I, others...
Conjunctions:
Particles
Numerals: one, two, three, first, second...

Assigning tags to words to the training data set is a manual work. Then we **build the algorithm and then we can tag them automatically.**

Main difficulties:
- Words with multiple possible tags: "Content" as noun, adjective, or verb
- New or unknown words

Approaches:
- **Rule-based taggers**: write manually some words to correct the mistakes. We need linguistics. Apply large set of rules. E.g. `If prev word is ADJ THEN word is N`
- **Stochastic taggers:** Use probabilities derived from large training corpus
- **Hybrid:** Induce rules from training corpus

5.2 Tagsets for English

Tagsets: Mainly from the **87-tag Brown corpus**. Others are 45-tag Penn Treebank tagset, and 61 tag C5 tagset.

The/**DT** grand/**JJ** jury/**NN** commented/**VBD** on/**IN** a/**DT** number/**NN** of/**IN** other/**JJ** topics/**NNS**./.

Although/**IN** preliminary/**JJ** findings/**NNS** were/**VBD** reported/**VBN** more/**RBR** than/**IN** a/**DT** year/**NN** ago/**IN** ,/, the/**DT** latest/**JJ** results/**NNS** appear/**VB** in today/**NN** 's/**POS** New/**NNP** England/**NNP** Journal/**NNP** of/**IN** Medicine/**NNP** ,/,

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	$
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	"	Left quote	' or "
POS	Possessive ending	*'s*	"	Right quote	' or "
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	[, (, {, <
PRP$	Possessive pronoun	*your, one's*	)	Right parenthesis	],), }, >
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	. ! ?
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	: ; ... – -
RP	Particle	*up, off*			

Figure 5.6: Penn Treebank POS tags

But there are some **challenges**, as it is hard to:

- Distinguish POS in a sentence:
  Mrs. Schafer never got **around/RP** to joining.
  All we got to do is go **around/IN** the corner.
  Chateau Petrus costs **around/RB** 250.

- Label the words that modify nouns (more in summary)
- Differentiate past participles from adjectives (more in summary)

5.3 Part of Speech Tagging

POS is the process of assigning a part of speech or other syntactic class marker to each word in a corpus.

Tokenization: Usually before POS. Separate commas, quotation marks...

POS is one of the many disambiguation tasks we will see in the book. The problem of POS-tagging is to **resolve** these ambiguities, choosing the proper tag for the context (**disambiguation**).

Tag	**Description**	***Example***
(	Opening parenthesis	*(,[*
)	Closing parenthesis	*),]*
*	Negator	*Not, n*t*
,	Comma	*,*
-	Dash	*-*
.	Sentence terminator	*.;?!*
:	Colon	*:*
ABL	Pre-qualifier	*Quite, rather, such*
ABN	Pre-quantifier	*Half, all*
ABX	Pre-quantifier, double conjunction	*Both*
AP	Post-determiner	*Many, next, several, last*
AT	Article	*A, the, an, no, a, every*
BE/BED/BEDZ/BEG/ BEM/BEN/BER/BEZ		*Be/were/was/being/am /been/arise/is*
C	Coordinating conjunction	*And, or, but, either, neither*
CD	Cardinal numeral	*Two, 2, 1962, million*
CS	Subordinating conjunction	*That, as, after, whether, before*
DO/DOD/DOZ		*Do, did, does*
DT	Singular determiner	*This, that*
DTI	Singular or plural determiner	*Some, any*
DTS	Plural determiner	*These, those, then*
DTX	Determiner, double conjunction	*Either, neither*
EX	Existential there	*There*
HV/HVD/HVG/HVN/HVZ		*Have, had, having, has*
IN	Preposition	*Of, in, for, by, to, on, at*
JJ	Adjective	
JJR	Comparative adjective	*Better, greater, higher, larger, lower*
JJS	Semantically superlative adj.	*Main, top, principal, chief, key, foremost*
JJT	Morphologically superlative adj.	*Best, greatest, highest, largest, latest*
MD	Modal auxiliary	*Would, will, can, could, may, must, should*
NN	(common) singular or mass noun	*Time, world, work, school, family, door*
NN$	Possessive singular common noun	*Father's, year's, city's, earth's*
NNS	Plural common noun	*Years, people, things, children, problems*
NNS$	Possessive plural noun	*Children's, artist's, parent's years'*
NP	Singular proper noun	*Kennedy, England, Rachel, Congress*
NP$	Possessive plural proper noun	*Plato's Faulkner's Viola's*
NPS	Plural proper noun	*Americans, Democrats, Chinese*

NPS$	Possessive plural proper noun	*Yankees', Gershwins' Earthmen's*
NR	Adverbial noun	*Home, west, tomorrow, Friday, North*
NR$	Possessive adverbial noun	*Today's, yesterday's, Sunday's, South's*
NRS	Plural adverbial noun	*Sundays, Fridays*
OD	Ordinal numeral	*Second, 2nd, twenty-first, mid-twentieth*
PN	Nominal pronoun	*One, something, nothing, anyone, none*
PN$	Possessive nominal pronoun	*One's, someone's, anyone's*

Most tagging algorithms are:

- **(5.4) Rule-based tagger**: they involve a large database of hand-written disambiguation rules, e.g *it is a noun rather than a verb if it follows a determiner.* **—> ex: EngCG**

- **(5.5) Stochastic taggers:** they resolve the tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context.) → ex: **HMM tagger.**

- **(5.6) Transformation-based tagger** or **Brill tagger**: (hybrid) shares features of both tagging architectures. It is based on rules that determine when ambiguous words should have a given tag, but it uses ML to automatically induce the rules from a previously tagged training corpus.

5.4 Rule-Based Part of Speech Tagging

The first stage used a dictionary to assign each word a list of potential parts-of-speech. The second stage used large lists of hand-written disambiguation rules to winnow down this list to a single part-of-speech for each word. In this section we describe a tagger based on this approach, the **EngCG** tagger (Voutilainen, EngCG 1995, 1999).

Example: p. 172

5.5 Hidden Markov Model (HMM) POS Tagging

Given a large annotated corpus of sentences: estimate probability distribution (probability of sequence of tags given sequence of words)

Maximize the (1) **probability of the sequence of tags** (bigram) and the (2) **probability of each tag generating a word.**

- **Stochastic** tagger algorithm.
- Uses the **Bayesian interference**.
- POS tagging is generally treated as a sequence **classification task**.

In Bayesian interference we start by considering all possible sequence classes – in this case, all possible sequences of tags. Out of this universe of tag sequences, we want to choose the tag sequence that is most probable given the observation sequence of words. *(^ means our estimate of the correct tag sequence)*

$$\hat{t_1^n} = \arg\max_{t_1^n} P(t_1^n | w_1^n)$$

However, with this expression above it is not clear how to start **making it operational**.

We transform it into another expression using the **Bayes' rule.** Bayes' rule gives us away to break down any conditional probability P(x|y) into three other probabilities:

$$\hat{t_1^n} = \arg\max_{t_1^n} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$$

Which is the same as:

$$\hat{t_1^n} = \arg\max_{t_1^n} P(w_1^n | t_1^n) P(t_1^n)$$

And under the Markov assumptions:

1. The probability of a word appearing depends only on its own POS tag; it is independent of other words around it and of the other tags around it.
2. The probability of a tag appearing is dependent only on the previous tag, rather than the entire sequence (bigram assumption).

$$\hat{t_1^n} \approx \arg\max_{t_1^n} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$$

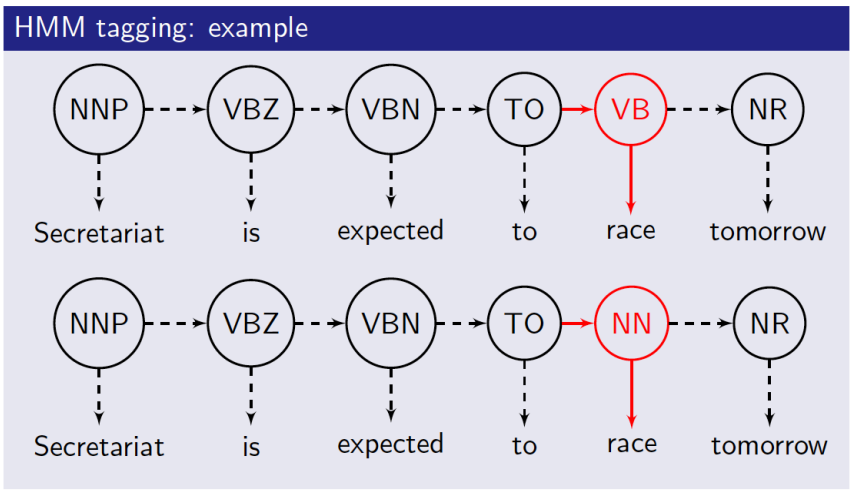
P(w | t) → *If we are expecting the third- person singular verb, how likely is that this verb will be "is"?*

P(t | t-1) → What is the P (NN|DT). We can count this by taking a corpus in which POS are labeled. How many times we see NN after the DT. (formula p. 175)

This is an oversimplification of HMM. We will introduce the (1) **deleted interpolation algorithm for smoothing these counts**, the (2) **trigram model of tag history**, and a (3) **model for unknown words.**

Before, we need to introduce the **decoding algorithm** by which these **probabilities are combined to choose the most likely tag sequence.**

5.5.1 Computing the Most Likely Tag Sequence: Example

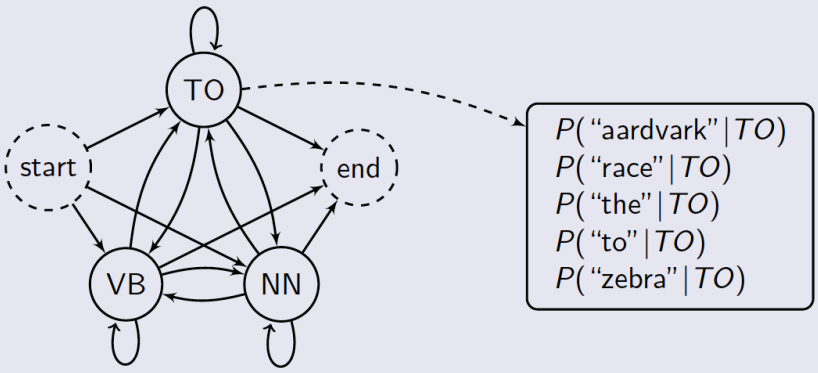


Each of these sentences has a prob.
Complete example p. 177 and 178

5.5.2 Formalizing Hidden Markov Model taggers

- **M'he quedat aqui. P. 178**

HMM as finite state machines



Example probabilities

	VB	TO	NN	PPSS
⟨s⟩	.019	.0043	.041	.067
VB	.0038	.035	.047	.007
TO	.83	0	.00047	0
NN	.004	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

Transition Probabilities

	I	want	to	race
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

Observation Probabilities

Viterbi

Given observations of length T, state-graph of length N
Create path probability matrix $viterbi[N+2, T]$
for each state s from 1 to N **do** {initialization}
  $viterbi[s, 1] \leftarrow a_{0,s} * b_S(o_1)$
  $backpointer[s, 1] \leftarrow 0$
end for
for each time step t from 2 to T **do** {recursion}
  for each state s from 1 to N **do**
    $viterbi[s, t] \leftarrow \max_{s'=1}^{N} viterbi[s', t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s, t] \leftarrow \arg\max_{s'=1}^{N} viterbi[s', t-1] * a_{s',s}$
  end for
end for
$viterbi[q_F, T] \leftarrow \max_{s=1}^{N} viterbi[s, T] * a_{s,q_F}$ {termination}
$backpointer[q_F, T] \leftarrow \arg\max_{s=1}^{N} viterbi[s, T] * a_{s,q_F}$
return backtrace path from $backpointer[q_F, T]$

Viterbi in words

- POS tagging is finding a path through a lattice
- The lattice describes all possible tag assignments
- For each node in the graph compute the best path to it
- Viterbi algorithm is a dynamic programming algorithm
  - Result of whole follows from computation of local situations

Unknown words

- What happens with words seen for the first time (testing)?

$$P(\text{Ishikawa}|\text{NN}) = 0???$$

- Solutions
  - Words with frequency 0 in training data behave similarly to those with frequency 1
  - Look at morphology of word
    - Ending in -s: likely to be plural noun
    - Ending in -ed: likely to be verb
    - Ending in -able: likely to be adjective

Unknown words

- Estimate probability of unseen word by combining morphological features

$$P(w_i|t_i) = P(\text{unknown\_word}|t_i) * P(\text{capital}|t_i) * P(\text{endings/hyph}|t_i)$$

Summary

- POS tags useful for higher-level language processing tasks
- Several approaches to automatic POS tagging exist
- Hidden Markov Models
  - trained from annotated training sentences
  - close to perfect ($\pm$97% accuracy) tagging performance

Hidden Markov Model (HMM) <—- we will see this in the machine translation chapter (mine!)
To calculate the text sequence
Given a large annotation corpus of sentences
We believe there are some patterns hidden in these data and we want to know them.
We want to know the prob of a text sequence given the sequence of words that I have.

W(1-n) —> sentence
T(1-n) —> text sequence (ex. preposition, conjugation)

We want to know the prob of a text sequence given a word sequence.

I don't care about the prob that much (number itself) but the text sequence as a result.

But the prob of a sentence to be in our corpus can be 0 (0 count)! So the assumption of Markov is that we don't need to have the full sentence in the corps but word by word and the sum the probabilities. The prob will be quite similar— we had the same problem last week.
El π enorme vol dir multiplicar totes les probabilities. - you need to do this word by word

Arg max —> I don't care about the actual prob, but I do want to have the highest!. Aquest es amb el que em quedo. -

We need to try all the combinations, which are crazy high. Computers are fast but it would take years anyway. There is a trick: HMM as a finite state machine
Viterbi algorithm —> this is also to reduce combinations. You are not going to calculate word by word separately, but you are going to reuse sequences that you have already calculated.

Nn nn nn
Nn nn vb
Nn nn to
Nn to nn

We can use the same nn nn calculated once and do not calculate it again. (PHOTO)

There is no dependency from c to a. The dependency depends on the bigram, trigram… From each node you want to have the best path.

K-fold contains the training set and the validation set. We left training set apart.

SYNTAXY – ARRANGEMENT OF WORDS

Chapter 12. **Formal Grammar of English / Context-Free Grammar**

(10/09/18)

We cover linguistics theory (terminology) and combine these words into larger linguistic structures. There is no unique way of doing so, but it is the way it fits linguistics models or theories.

One part of the chapter is about a particular linguistic theory and the other part is more a math theory.

Then we have these final state machine and we can proof that they are less powerful than a context free grammar, just CFG can say more. We have to distinguish linguistics from math's.

Syntax: refers to the way in which words are arranged together. How to build words together.

- **Constituency**: words grouped together and if you change the order they lose the meaning and have essentially the same function. Idea that groups of words may behave as a single unit or phrase (constituent). An example are **Noun phrases** (sequence of words surrounding at least one noun - *Harry the Horse, Tilburg Univeristy*) – we will be able to model these constituents with the CFG

- **Grammatical relations:** formalization of ideas from traditional grammar (eg. SUBJECT, OBJECT). They refer to functional relationships between constituents in a clause.

- **Subcategorization** and **Dependency**: refer to a certain kind of relations between words and phrases.

  Example 1: the verb *want* can be followed by:

  - An infinitive (e.g. *I want **to fly** to Detroit)*
  - A noun phrase (e.g. *I want **a flight** to Detroit)*

  Example 2: the verb *find* **cannot** be followed by an infinitive (e.g. **I found **to fly** to Detroit)*

  Facts about the **subcategorization** of the verb

  This kind of mechanism can be modeled by grammars that are based on **context-free grammars**, which are fundamental for computer application such as *semantic interpretation*, *dialogue understanding* or *grammar checkers.*

Context-Free Grammar (CFG)

It is the most commonly used **mathematical system for modeling constituent** structure in English and other natural languages.

It is the mathematical perspective. We want to have a formalism that tells the computer "this is how it should be" in a way that computers understand. Mathematical idea to describe English.

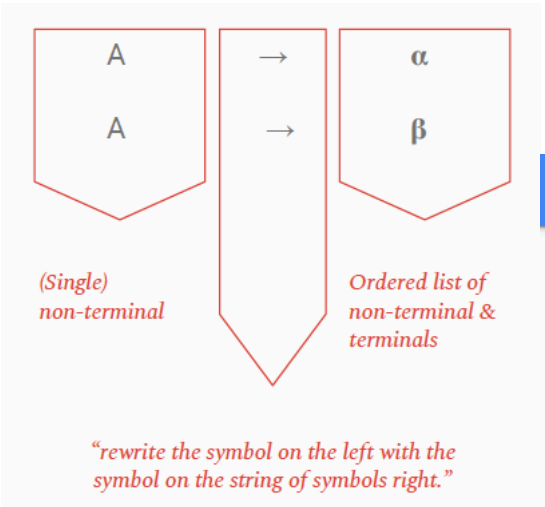
A CFG consists of:

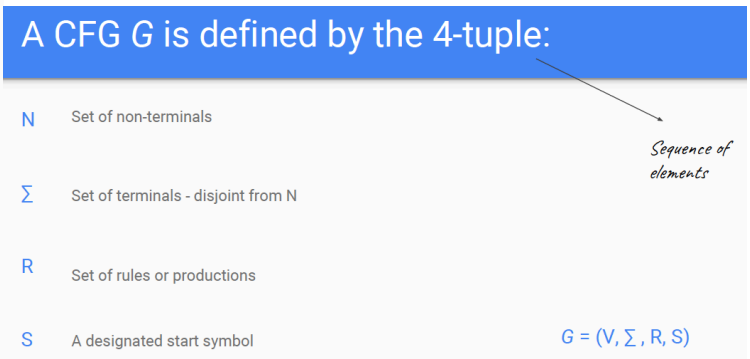
- A set of **rules** (or **productions**), each expressing the ways in which symbols of the language can be grouped and ordered together. The rules can **be hierarchically embedded**. (No-terminal are NP, Det, Noun…)
  - Noun Phrase (NP) → it can be a determinal (ei. the) or a nominal (ei. man)

$$NP \rightarrow Det\ Nominal$$
$$NP \rightarrow ProperNoun$$
$$NP \rightarrow Noun \mid Noun\ Nominal$$

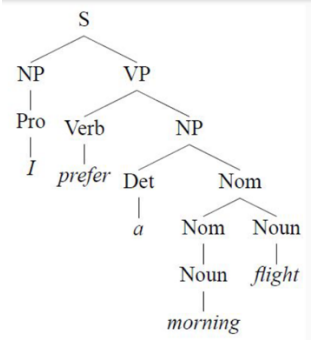
- A **lexicon (lexicalized rules)** of words and symbols. (terminals are the words)
  - Noun → women
  - Verb → runs | sees | …
  - Det → the | a | this

$$Det \rightarrow a$$
$$Det \rightarrow the$$
$$Noun \rightarrow flight$$





Noun	Flights | breeze | trip | morning
Verb	Is | prefer | like | need | want | fly
Adjective	Cheapest | non-stop | first | latest
Pronoun	Me | I | you | it
Proper-Noun	Alaska | Baltimore | Los Angeles | united | American
Determiner	The | a | an | this | these | that
Preposition	From | to | on | near
Conjunction	And | or | but



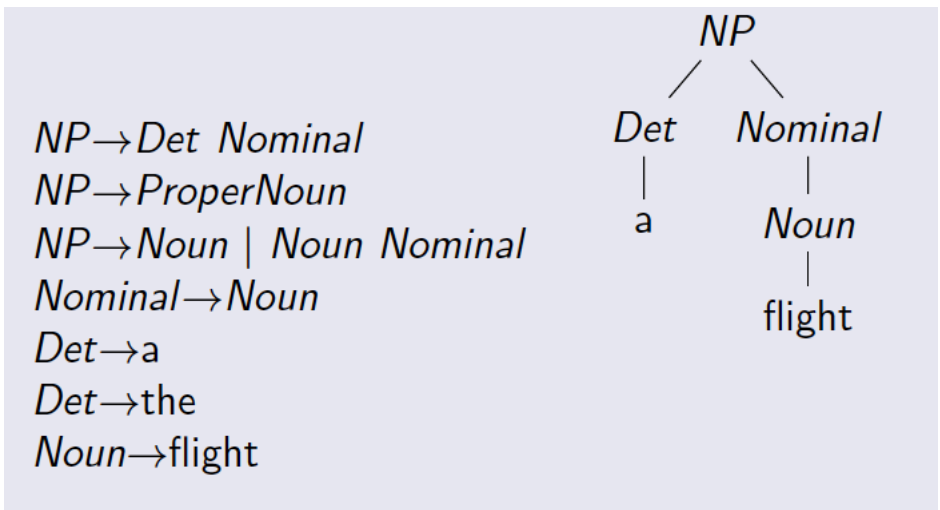
Bracketed Notation: a more compact way to represent parse trees (Figure 5).

[$_S$ [$_{NP}$ [$_{Pro}$ I]] [$_{VP}$ [$_V$ prefer] [$_{NP}$ [$_{Det}$ a] [$_{Nom}$ [$_N$ morning] [$_{Nom}$ [$_N$ flight]]]]]]

After rules written… (we have the power to write them)

We have 2 choices. We can either start from the sentence. It is a valid sentence if I can find a structure that validates it.

Ex. I start with the sentence "*A flight"* and I ask if this is a Noun Phrase. (check image below). Then I start bottom up and look at my lexicon. I see that A can be a *det*. Then flight says that it is a noun. We don't have any rules that say that *det* and *noun* are something… but Noun is a nominal! And now we have a rule that if we have a det and nominal they can be combined as a NP. And this is the last step. It is a valid NP. This is called **Derivation / parse tree.**



It is **called Context-Free** because rules can be applied regarding of the context. For example, NP will always be Det Nominal, whatever context NP occurs in.

The second choice is from the top down. We want to generate a sentence. Look at the grammar you have (rules) and check which terminals can we write from the non-terminals.

Sentence Level Constructions

1. **Declarative structure:** have a **subject noun phrase** followed by a **verb phrase**.
   Example: “I (S) prefer (VP) a morning flight”

2. **Imperative structure:** often begin with a **verb phrase** and have no subject. Usually used for commands and suggestions.
   Example: “Show (VP) the lowest fare”

3. **Yes-no question structure:** often used to ask questions; they begin with an auxiliary verb, followed by a subject *NP*, followed by a *VP*.
   S → Aux NP VP
   Example: “Does (Auxiliar) this flight (S) stop (VP) in Paris?”
4. **Wh-structures:** one of its constituents is a wh-phrase (*who, when, where, what, which, how, why)*. The wh-structure is identical to the declarative structure, except that the first noun phrase contains some wh-word.
   S → Wh – NP VP
   Example: “What (wh) airlines (NP) fly (VP) from Burbank to Denver?” ·

5. Rule: in the **wh-non-subject-question** structure, the wh-phrase is not the subject of the sentence, and so the sentence includes another subject. The auxiliary appears before the subject *NP*, just as in the yes-no question structures.
   S → Wh-NP Aux NP VP
   Example: “What flights do you have from Burbank to Tacoma Washington?”

   These constructions are called **long-distance dependencies** because the *Wh-NP what flights* is far away from the predicate that it is semantically related to (which is the main verb *have* in the *VP*).

   > In some models this is seen as a semantic relation. In some models of parsing however, this is considered a syntactic relation and the grammar is modified to insert a small marker called a **trace** or **empty category** after the verb.

Noun phrases

[More about Clause and Sentences a) Independent clauses b) Dependent clauses c) Relative clauses d) Noun clauses] – IMPRIMIR EL PPT

Noun phrases

A head (noun) with modifiers

- Prenominal modifiers
  - Determiners: **a** flight
  - Cardinal numbers: **two** friends
  - Adjective phrases: the **least expensive** fare
- Postnominal modifiers
  - Prepositional phrases: all flights **from Cleveland**
  - Non-finite clauses: any flights **arriving after eleven a.m.**
  - Relative clauses: a flight **that serves breakfast**

Noun phrases

CFG rules (incomplete)

- Prenominal modifiers
  $NP \rightarrow (Det)\ (Card)\ (Ord)\ (Quant)\ (AP)\ Nominal$
- Postnominal modifiers: prepositional phrases
  $Nominal \rightarrow Nominal\ PP\ (PP)\ (PP)$
- Postnominal modifiers: non-finite clauses
  $Nominal \rightarrow Nominal\ RelClause$
  $RelClause \rightarrow$ (who | that) VP

Agreement

We have to have a rule that one part of the tree influence the other. Then we need some rules for the red personal singular. Therefore, the grammar gets twice as large:

S —> $NP_{\text{3rd personal sing}}$ VP3rd personal sing
S —> NP non 3rd personal sing VP non 3rd personal sing
NP 3rd pers sing —> Nom 3d pers sing
NP non 3rd personal sing —> Nom non 3rd personal sing

S —> NPx VPx —> this is easier to write down. So is says that x has to match x. These are diff formalism, and there are a big choice of them.

3sg (third-person singular) Ex: Does
Non-3sg: Ex: Do

Add rules to the lexicon:

3sgAux → does | has | can | ...
Non3sgAux →do | have | can | ...

$$S \rightarrow Aux\ NP\ VP$$

$S \rightarrow 3sgAux\ NP\ VP$ or $S \rightarrow non3sgAux\ NP\ VP$

English pronouns have nominative and accusative versions). Another multiple to the rule set is added when a language has **gender agreement**.

Agreement in CFGs

S	→ *3sgAux 3sgNP VP*
S	→ *Non3sgAux Non3sgNP VP*
3sgAux	→ does | has | ...
Non3sgAux	→ do | have | ...
3sgNP	→ *Det SgNominal*
Non3sgNP	→ *Det PlNominal*
SgNominal	→ *SgNoun*
PlNominal	→ *PlNoun*
SgNoun	→ flight | fare | ...
PlNoun	→ flights | fares | ...

The Verb Phrases and Subcategorization

Verb phrases

Verb phrases consist of a verb and optional complements

VP → *Verb*	disappear
VP → *Verb NP*	prefer a morning flight
VP → *Verb NP PP*	leave Boston in the morning
VP → *Verb PP*	leaving on Thursday

Entire sentences can follow a verb (sentential complements)
You $[_{VP}$ $[_{V}$ said] $[_{S}$ there were two flights that were the cheapest]]
VP → *Verb S*

There are 2 subcategories:

1. Transitive: verbs that take a direct object NP (ex. Find)
2. Intransitive: verbs that don't take a direct object NP (ex. Disappear)

A verb such as Find subcategorizes for an NP, and a Verb like Want subcategorizes for either an NP or a non-infinite VP.

Sub-categorization does not affect the grammar rules.

Subcategorization frames

Frame	Verb	Example
$\emptyset$	eat, sleep	I want to walk
NP	prefer, leave	Find [$_{NP}$ your keys]
NP NP	give	Show [$_{NP}$ me] [$_{NP}$ that]
PP-from PP-to	travel	Fly [$_{PP\text{-}from}$ from A] [$_{PP\text{-}to}$ to B]
NP PP-with	load	Help [$_{NP}$ me] [$_{PP\text{-}with}$ with that]
...		

(no entenc molt bé aquesta diapo)

Subcategorization in CFGs

- Same solution as with agreement
- Explosion of number of rules
- Solution: Use feature structures

Auxiliaries – Helping verbs

Aux Type	Verb	Subcategorizes for a	Example in use
Modal	*can, could, may, might, must, will, would, shall, should*	VP whose head is a bare stem	*will try to find a flight*
Perfect	*have*	VP whose head verb is in past participle	*have booked 3 flights*
Progressive	*be*	VP whose head verb is in the gerundive participle	*am going from Atlanta*
Passive	*be*	VP whose head verb is in past participle	*was delayed by weather*

Coordination – can conjoin the major phrases types to form larger constructions of the same type: and, or, but.
NP → NP and NP
VP → VP and VP

Treebanks – Més info al resum de la lectura

A corpus in in which every sentence is syntactically annotated with a parse tree (context free grammar rules can be used to assign a parse tree to any sentence).

Sentences represented in a tree-bank implicitly constitute a grammar of the language.
● The Grammar used to parse the Penn Treebank is relatively flat, which means that there are often very long rules.
● This creates many problems in probabilistic parsing algorithms. Therefore, its common to make various modifications to a grammar extracted from a treebank.

tgrep/TGrep2 are exploratory tools that are used to extract tree structures based on the input pattern
● A pattern in tgrep/Tgrep2 consists of a specification of a node and its relationship to other nodes.
● A node specification can then be used to return the subtree rooted at that node.

● tgrep/Tgrep2 can also specify the information about the links
Example:
The operator "**<**" means that any NP node that immediately dominates a PP:
NP < PP
This matches an NP that dominates a PP and is immediately followed by a VP:
NP << PP . VP

Grammar Equivalence and Normal Form

A formal language is defined as a set strings of words. So, this suggest that we could ask if two grammars are equivalent by asking if they generate the same set of strings. In fact, it is possible to have two distinct context-free grammars generate the same language.

Two kinds of equivalence:

> **Weak equivalence**: If two grammars generate the same set of strings but do not assign the same phrase structure to each sentence
>
> **Strong equivalence**: They generate the same set of strings and they assign the same phrase structure to each sentence (allowing merely for renaming non-terminal symbols)

Normal forms: each of the productions take a particular form

Example:

Context-free grammar is in **Chomsky Normal Form (CNF)** is it's ε-free and if in addition each production is either of the form **A → B C** or **A → a**. That is, the right-hand side of each rule production is either has two-non-terminal symbols or one terminal symbol.

Chomsky normal form are **binary branching** → have binary trees (down to the prelexical nodes)
Any grammar can be converted into CNF, such as:

A → B C D

converted into two CNF rules become:

A → B X
X → C D

Binary branching from CNF can actually produce smaller grammars, such as:

VP → VBD NP PP*

To:

VP → VBD PP
VP → VP PP

Finite-State and Context-Free Grammars – No ho entenc

Why context-free grammars is more useful in this part than finite-state methods?

There are two reasons:

1. Mathematical:
    a. Certain syntactic structures present in English (and other natural languages) make them not regular languages.
    b. Because of **regular grammars**: equivalent alternative to finite-state machines and regular expressions for describing regular languages.
    c. The rules in regular grammar are a restricted form of the rules in CFG because they are in right-linear or left-linear form.
        i. Right linear such as A -> w* or A -> w*B
    d. These rules can't express recursive **center-embedding** (a non-terminal is rewritten as itself) rules, such as:

$$A \Rightarrow^{*} \alpha A \beta$$

2. Finite-state methods might able to deal with syntactic facts in question, but they often don't express them in ways that make generalizations obvious, which lead to understandable formalism, or produce structures od immediate use in subsequent semantic processing.

In conclusion, a language can be generated by a finite-state machine if and only if the grammar that generates *L* does not have any **center-embedded** recursions of this form.

Center-embedding rules are needed to deal with artificial problems such as the language $a^n b^n$, or for practical problems such as checking for correctly matching delimiters in programming and markup languages.

Finite-state methods would be sufficient when there are many practical purposes where matching syntactic and semantic rules aren't necessary.

EXERCISE:

Check if it is a valid sentence. 1rs thing, we need a grammar to check on. -PHOTO

A woman runs
S
NP VP
Det Nom
N
A women runs

But imagine we have

A women (runs) a women

In this last case the verb *runs* do not fit it terms of meaning, but it is in our rules…
So, we will have now three types of verbs, therefore, we will have to write three types of rules for VP. An example would be:

VP —> V' NP
VP —> V
V' —> jdnafjkds whatever

Grammar

Lexicon

Noun	→woman | man | →binoculars ...
Verb	→runs | sees | ...
Adjective	→tall | green | ...
Pronoun	→I | you | me | ...
Proper-Noun	→Mary | John | ...
Determiner	→the | a | this | ...
Preposition	→in | with | ...
Conjunction	→and | or | but | ...

Rules

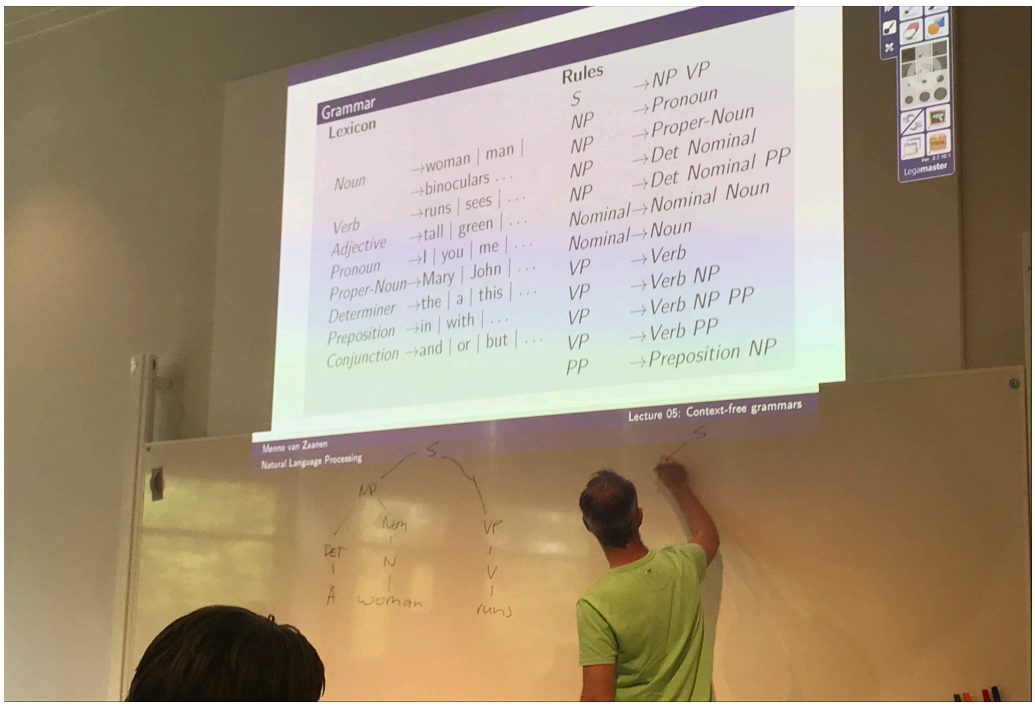
S	→*NP VP*
NP	→*Pronoun*
NP	→*Proper-Noun*
NP	→*Det Nominal*
NP	→*Det Nominal PP*
Nominal	→*Nominal Noun*
Nominal	→*Noun*
VP	→*Verb*
VP	→*Verb NP*
VP	→*Verb NP PP*
VP	→*Verb PP*
PP	→*Preposition NP*

Automatic parsing
We will see them in the next chapter, but we already have some algorithms that do the parsing automatically.

a^n b^n It is a sentence with the same number of a than the b.

Fine-state machine are really efficient! It takes less time than context - free parsing. That is why we still use it (double check this)

Context free grammar is more powerful than fine-state machine, because fine-state machine has not memory.



Chapter 13. **Parsing with Context-Free Grammars**

13/09/18

Parsing try all the grammar with all the combinations till they find a proper way to describe the sentence.

Syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure to it. Useful in:

- Grammar checking: if a sentence cannot be parsed, it may contain errors
- Semantic analysis: it may play an important role in applications like question answering and information extraction.

Top-down parsing: try all possibilities. Trees are grown downward until they eventually reach the part-of-speech categories at the bottom of the tree.

Bottom-up: The parse is successful if the parser succeeds in building a tree rooted in the start symbol S that covers all of the input. So, it looks up every word's POS at the beginning and builds up different trees based on the amount of ambiguous POS.

One vs the other:

Top-down:

- Never wastes time exploring trees that cannot result in an S. Only generates (sub)trees that have S as a root
- Never explores subtrees that cannot find a place in some S-rooted tree
- Spends time on trees that are not consistent with the input (inefficient)

Bottom-up:

- Trees that have no hope of leading to an S are still generated, which is inefficient
- Suggest trees that are at least locally grounded in the input, which is a good thing

13.2 Ambiguity

We have seen POS ambiguity, and POS disambiguation. New type of ambiguity: structural ambiguity.

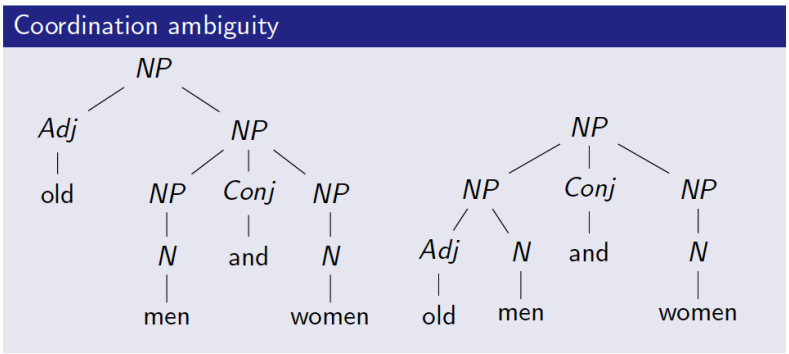
Structural ambiguity arises in the syntactic structures used in parsing. It happens when the grammar assigns more than one possible parse to a sentence. Two common kinds of this kind of ambiguity are:

- **Attachment ambiguity**. This happens when a particular constituent can be attached to the parse tree at more than one place. So, a part of the sentence can belong (or describe) more

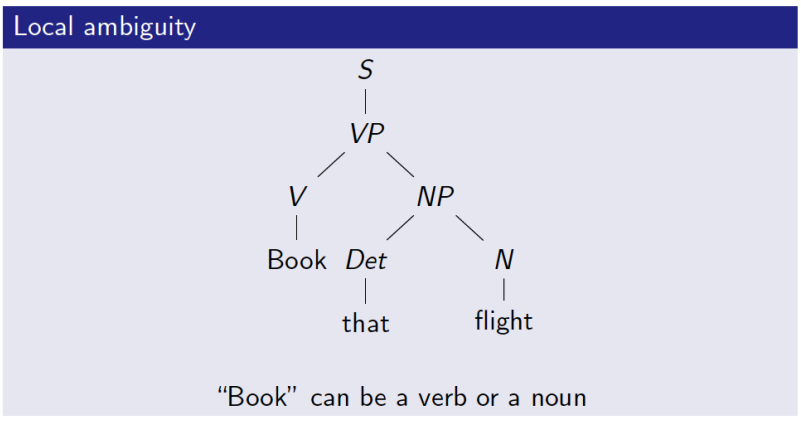
parts of a sentence. For example: “We saw the Eiffel Tower flying to Paris”. The Eiffel Tower can belong to flying (in this case, the Eiffel Tower flies), or to saw, in which case we saw the Eiffel Tower when we flew to Paris. Which makes more sense.

- **Coordination ambiguity**. Different sets of phrases can be conjoined by a conjunction like and. For example: old men and women: [old [men and women]], or [old men] and [women].

Ultimately, most natural language processing systems need to be able to choose the correct parse from the multitude of possible parses through a process known as **syntactic disambiguation**. To do this, a system will need statistical, semantic and pragmatic knowledge. If it doesn’t have this knowledge, it has to return all possible parse trees for a given input. This may lead to lots of different options, also because the number of parses grows exponentially at the same rate as the number of parenthesizations of arithmetic expressions.



And even if a sentence is not ambiguous, it can be inefficient to parse because of **local ambiguity**, which occurs when some part of the sentence is ambiguous (so when it has more than one parse), even if the whole sentence is not ambiguous.
Example: the whole sentence ‘Book that flight’ is not ambiguous, but when the parser sees the word Book, it cannot know if the word is a verb or a noun until later.



13.3 Search in the Face of Ambiguity

If we want to build up all possible parse trees at the same time (as described in top-down parsing), this will require a great amount of memory. An alternative approach is to use an **agenda-based backtracking strategy**. A backtracking approach expands the search space by systematically

exploring one state at a time. When the given strategy arrives on a point where the tree does not match the input, the search continues by returning to an unexplored option already on the agenda. However, this can also lead to failure because not all of the input is covered. They also need to go over the same part multiple times, which causes wasted effort in both top-down and bottom-up parsing.

Resolving ambiguity:

- The **local ambiguity** is solved by the parsing algorithm
- The **Structural Ambiguity** cannot be resolved by the parsing algorithm. It returns all possible parses. It is normally resolved using external knowledge:
  - Statistical Knowledge
  - Semantic Knowledge
  - Pragmatic Knowledge

Solution:

13.4 Dynamic Programming Parsing Methods

Dynamic programming systematically fills in tables of solutions to sub-problems (same as Minimal Edit Distance). When complete, the tables contain the solution to all the sub-problems needed to solve the problem as a whole. In the case of parsing, such tables store subtrees that have been discovered once. The efficiency rises because these subtrees do not have to be discovered multiple times. They try to write down all the **possibilities parse trees in an efficient ways.** If we write them all down they take a huge amount of space.

The three most widely used parsing methods are:

1. **Cocke-Kasami-Younger (CKY) algoritm** → must be in CNF
2. **Earley algorithm**
3. **Chart parsing**

CYK Algorithm

Major requirement: The grammar used with it must be in Chomsky Normal Form (CNF).
Bottom - Up

Convert it to CNF:
In Chompky we just have 2 grammar rules:

- A —> B C (A, B, C are non-terminal - always in capital letters)
- A —> w (w is a terminal - always in lower letters)

Conversion to CNF

- If a rule is already in CNF, don't do anything
- If a rule has both terminals and non-terminals, rewrite
  - *INF-VP*→to *VP*
  - *INF-VP*→*TO VP*
    TO→to
- If a rule has a single non-terminal, replace in other rules
  Replace occurrences "middle" non-terminal with right-hand sides
  - *NP*→*Noun*
    Noun→house
    Noun→table
  - *NP*→house | table
- If a rule has more than two elements, split rule
  - *VP*→*Verb NP PP*
  - *VP*→*X1 PP*
    X1→*Verb NP*

Exercice:

Convert grammar to CNF: E (starting symbol and stands for expression).

E —> E + T E —> T T —> T * F T —> T / F T —> F F —> num F —> id	E —> E **P** T —> E P E P —> + ~~T —> T M F~~ M —> * ~~T —> T D F~~ D —> / ~~E —> T~~ ~~T —> F~~ ~~F~~ —> num —> E ~~F~~ —> id —> E	E —> T M F —> E M F —> E D E E —> T D F —> E D F —> E D E E —> F —> therefore, all F can be E. Eliminate the F.	P —> + M —> * D —> / E —> X1 E X1 —> E P E —> num E —> id E —> X2 E X2 —> E M E —> X3 E X3 —> E D
They are all wrong except the last 2. + is a terminal. We have to convert it into a non terminal	But! If there is a single non terminal, replace in other rules. I am going to get rid of T, because it can be replaced with E	But, if the rule has more than 3 elements, split rule —>	Now we have everything in CNF

How does CYK Algorithm works?

$_0a_1b_2c_3d_4$

Now imagine a grammar:

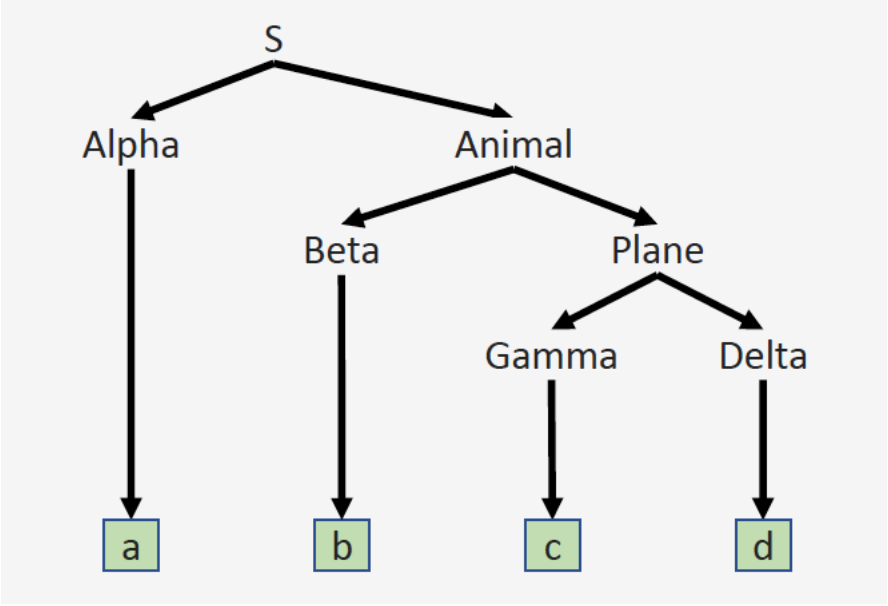
• S → Alpha Animal
• Animal → Beta Plane
• Plane → Gamma Delta

Also imagine a Lexicon:
• Alpha → a
• Beta → b
• Gamma → c
• Delta → b

0 a 1	b 2	c 3	d 4
Alpha→ a R [0, 1]	- R [0, 2]	- R [0, 3]	S→ Alpha Animal R [0, 4]
	Beta → b R [1, 2]	- R [1, 3]	Animal → Beta Plane R [1, 4]
		Gamma → c R [2, 3]	Plane → Gamma Delta R [2, 4]
			Delta → d R [3, 4]

- Grammar:
  - S → Alpha Animal
  - Animal → Beta Plane
  - Plane → Gamma Delta
- Lexicon:
  - Alpha → a
  - Beta → b
  - Gamma → c
  - Delta → d

R stands for range in the matrix. Our green



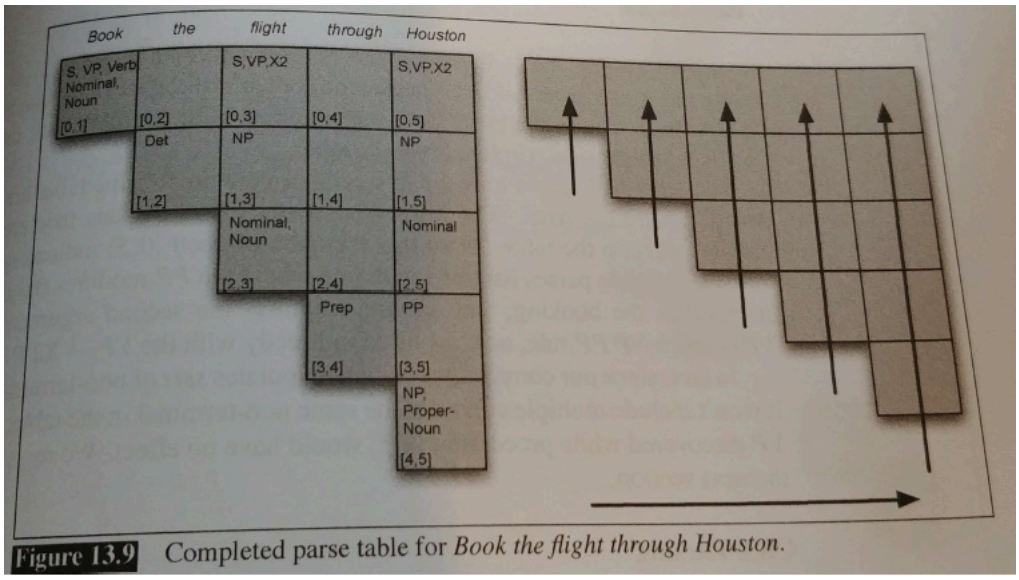


Figure 13.9 Completed parse table for *Book the flight through Houston.*

CKY algorithm(*words*, *grammar*) returns *table*

for j from 1 to $|words|$ **do**
  $table[j-1, j] \leftarrow \{A | A \rightarrow words[j] \in grammar\}$
  for i from $j-2$ downto 0 **do**
    for k from $i+1$ to $j-1$ **do**
      $table[i,j] \leftarrow table[i,j] \cup \{A | A \rightarrow BC \in grammar, B \in table[i,k], C \in table[k,j]\}$
    end for
  end for
end for

Early Algorithm – Assignment done

Top-down

The PREDICTOR and COMPLETER operators add states to the chart entry being processed and SCANNER adds a state to the next chart entry.

Earley algorithm(*words*, *grammar*) returns *chart*

```
ENQUEUE(γ → •S, [0,0]), chart[0])
for i from 0 to |words| do
  for each state in chart[i] do
    if COMPLETE?(state) then
      COMPLETER(state)
    else
      if NEXTCAT(state) is part-of-speech then
        SCANNER(state)
      else
        PREDICTOR(state)
      end if
    end if
  end for
end for
```

ENQUEUE(*state*, *chart_entry*)

```
if state not already in chart_entry then
  PUSH(state, chart_entry)
end if
```

COMPLETER($(B \rightarrow \gamma\bullet, [j,k])$)

```
for each (A → α • Bβ, [i,j]) in chart[j] do
  ENQUEUE((A → αB • β, [i,k]), chart[k])
end for
```

SCANNER($(A \rightarrow \alpha \bullet B\beta, [i,j])$)

```
if B ⊂ PARTS_OF_SPEECH(word[j]) then
  ENQUEUE((B → word[j]•, [j, j+1]), chart[j+1])
end if
```

PREDICTOR($(A \rightarrow \alpha \bullet B\beta, [i,j])$)

```
for each (B → γ) in GRAMMAR_RULES_FOR(B, grammar) do
  ENQUEUE((B → •γ, [j,j]), chart[j])
end for
```

Chart Parsing

13.5 Partial Parsing

There are 2 possible approaches: Finite State Transducers and Chunking.

Chunking: - més al resum dels companys

Parsing is not hard, but writing a grammar is really hard. You can be writing the grammar for years and still be missing parts. So, do we really want to have the full grammar? Not really, we do only need to have chunks /small parts → limited set of rules that if you apply them to your sentences you do not get the full thing but **only the interesting parts**.

Ex: "*an old man walks*" // an old man → NP (noun phrase). *Walks* is outside the NP
I, O, B (in, out, begin) An (BNP) / old (INP) / man (INP) / walks (O)

If we are interested in the VP as well, walks will be the BVP, the beginning of the VP
With these tags we can indicate if the words are inside the whole block or outside.

An old man walks a book —> a book is a now NP, therefore, a is BNP and book is INP

Finite-State rule-based chunking – *als apunts dels companys*

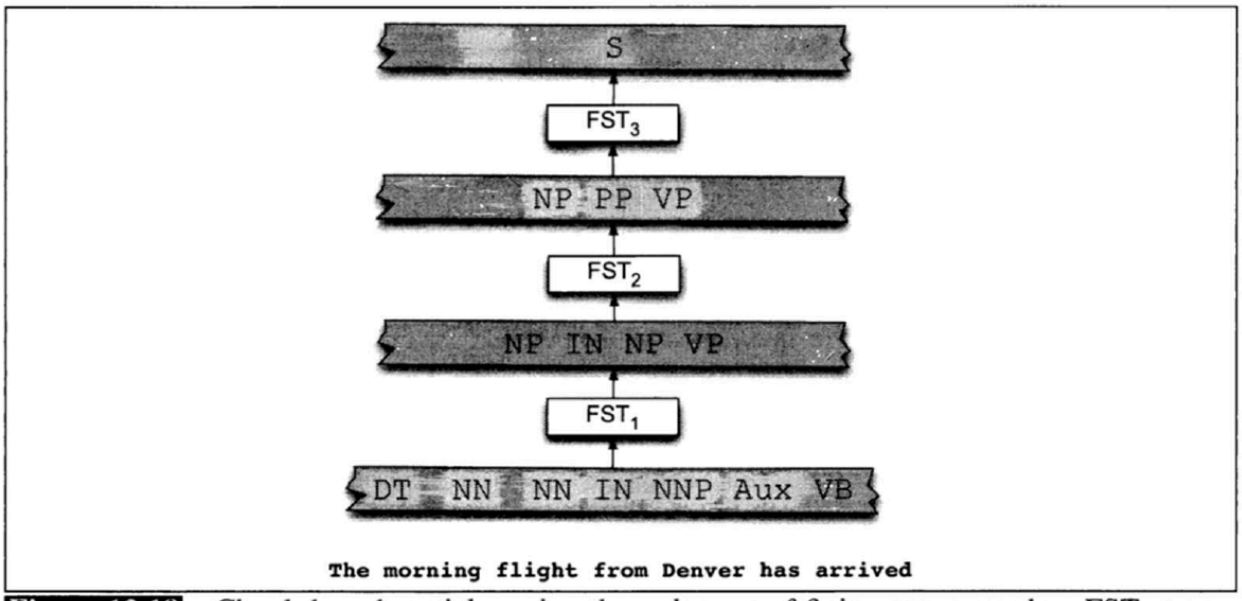


Figure 13.18 Chunk-based partial parsing through a set of finite-state cascades. FST_1 transduces from part-of-speech tags to base noun phrases and verb phrases. FST_2 finds prepositional phrases. Finally, FST_3 detects sentences.

Machine Learning based approaches on chunking - *als apunts dels companys*

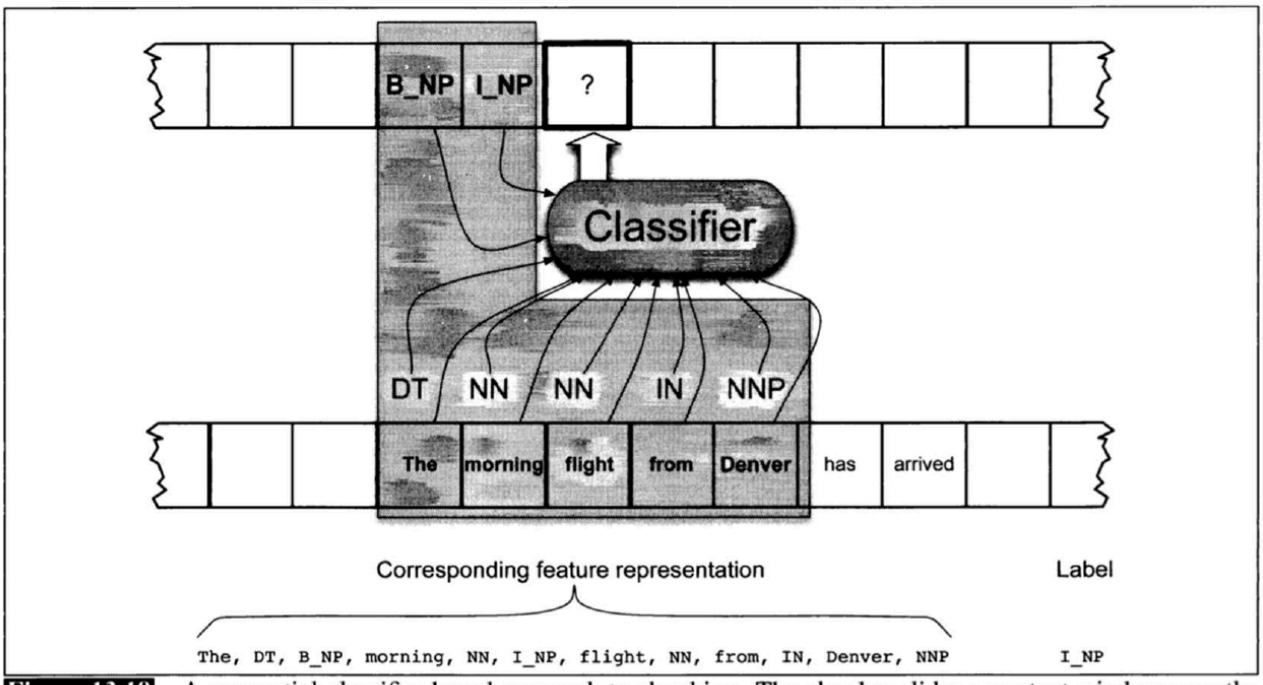


Figure 13.19 A sequential-classifier-based approach to chunking. The chunker slides a context window over the sentence, classifying words as it proceeds. At this point, the classifier is attempting to label *flight*. Features derived from the context typically include the words, part-of-speech tags as well as the previously assigned chunk tags.

Classifier: something that receives a particular input and assigns a tag.

During training, the classifier would be provided with a training vector consisting of the values of 13 features: the two words to the left of the decision point, their parts-of-speech and chunk tags, the word to be tagged along with its part-of-speech, the two words that follow along with their part-of speech, and finally the correct chunk tags.

Chunking-system evaluation - *als apunts dels companys*

- Precision
- Recall
- F Measure

Ambiguity:

If your grammar contains the rules that allow you to do two diff analysis of the same sentence, then your grammar is ambiguous. This is an issue. Which is the analysis that we normally see? Another problem that we are not solving here. So, the parses solve all possible analysis.

Grammar1

$S \rightarrow AB\ C$
$AB \rightarrow A\ B$
$A \rightarrow$ a
$B \rightarrow$ b
$C \rightarrow$ c

Input sentence: "a b c"
Unambiguous

Grammar2

$S \rightarrow AB\ C$
$S \rightarrow A\ BC$
$AB \rightarrow A\ B$
$BC \rightarrow B\ C$
$A \rightarrow$ a
$B \rightarrow$ b
$C \rightarrow$ c

Input sentence: "a b c"
Ambiguous

SEMANTICS AND PRAGMATICS – MEANING

Chapter 17: **The Representation of Meaning**

17/09/2018

Sentences are written.
Utterances are spoken.
Syntactic processing extracts linguistic information
What does the writer/speaker want us to read/hear?
What is the meaning of a sentence / utterance?

Semantic analysis: Extracts meaning from natural language. The process whereby such representations are created and assigned to linguistic inputs.

The output of semantic analysis should be in format that:

- can be dealt with by a computer
- enables us to do what we want to do (answering questions about a text, summarizing a text, translating a text...)

Meaning representation Language. Having access to the phonological, morphological and syntactic representation that we have seen will not help us in the meaning. Specify syntax and semantics of representation of meaning. They do no tell us how to produce meaning of a sentence.

We need a representation that links the linguistics elements to the non-linguistic knowledge of the world. Semantics is the link.

They compare:

- Regular expressions vs finite state automata
- Context-free grammars vs parsing
- Meaning representation vs semantic analysis

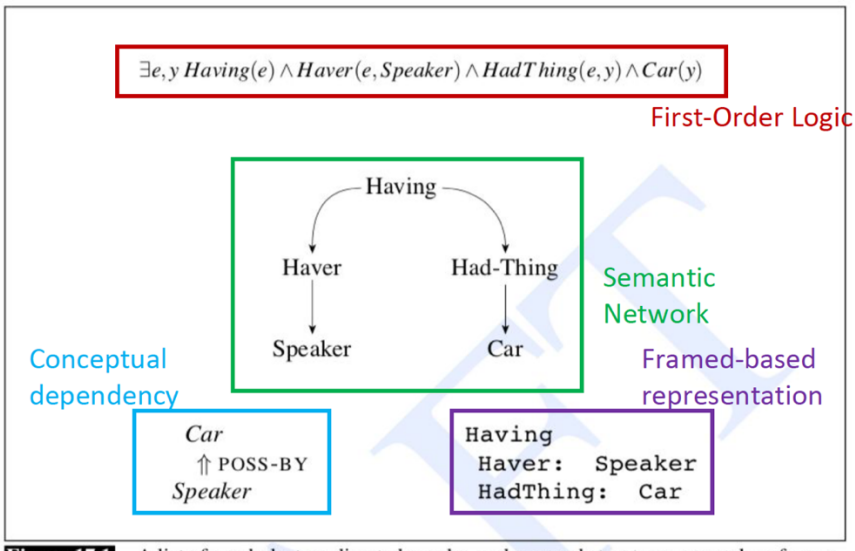


Figure 17.1 A list of symbols, two directed graphs, and a record structure: a sampler of meaning representations for *I have a car.*

17.1 Computacional Desiderata for Representations (requirements for meaning representations)

Verifiability: Relating meaning of sentence to the world. Systems' ability to compare the state of affair in some world as modeled in a knowledge base.
Ex: *Does Maharani serve vegetarian food?*

Unambiguous representation
Sentences may be ambiguous, semantics may not. Representations should represent ambiguities unambiguously. We want to be able what the sentence means. If it has serval meanings, we want to describe all of them in the representation.

Canonical form
Different sentences with same meaning → same representation
Some words in these sentences may have various **word sense** and that some of the senses are synonymous with one another. **Word sense disambiguation:** choosing the right sense in context.

Inference:
Draw conclusions based on:

- Meaning representation of sentence
- Knowledge base (e.g word knowledge)

Ex: *Can vegetarians eat at Maharani? // I'd like to find a restaurant where I can get vegetarian food.*

Expressiveness:

Representation deals with wide range of meanings.
Expressive enough to handle an extremely wide range of subject matter. FOL is expressive enough to handle quite of what needs to be represented.
It also depends on the application. We want to express everything we need in a domain.

Predicate-argument structure:

We can describe the meaning in diff ways. This is a choice we need to make. You can designed however you like, as long as it is clear.

- **Verb** as a predicate: For example, if we want to have the verb as a predicate (verb as the important thing):
  - *I* (arg1) *want* (predicate) *(Italian Food)* (arg2) → Want (I, ItalianFood)

- **Preposition** as a predicate:
  - An (Italian restaurant) (arg1) under (predicate) (fifteen dollars) (arg2)
- **Nouns** as predicates:
  - Make a reservation (predicate) for (this event) (arg2) for a (table for two people) (arg 4) at (8) (arg3)
  - Note: Implicit argument (arg1): Hearer

Want (I, ItalianFood)
Want (. , .)
Want (. , . , .) —> *what goes inside are the **arguments**. **Predicates** are the ones that are before the ( ). Predicates always return **True** or **False**.*

17.2 Model Theoretic Semantics

Most meaning representation schemes have the ability to represent objects, their properties, and various relations among them.

Expressions in a meaning representation language can then be mapped in a systematic way to the elements of the model - this constitutes the necessary bridge between formal representations and representations in the knowledge base.

Non-logical vocabulary: open-ended set of names for objects, properties and relations in the world. → they have a **denotation (**ex. Object)**. Domain** is the set of objects. **Extensional approach:** when relations among objects are denoted as sets of ordered lists or tuples of domain elements.

Logical vocabulary: closed set of symbols, operators, quantifiers, links... that provide the formal means for composing expressions.

Domain	$\mathscr{D} = \{a,b,c,d,e,f,g,h,i,j\}$
Matthew, Franco, Katie and Caroline	a,b,c,d
Frasca, Med, Rio	e,f,g
Italian, Mexican, Eclectic	h,i,j
Properties	
Noisy	$Noisy = \{e,f,g\}$
Frasca, Med and Rio are noisy	
Relations	
Likes	$Likes = \{\langle a,f\rangle, \langle c,f\rangle, \langle c,g\rangle, \langle b,e\rangle, \langle d,f\rangle, \langle d,g\rangle\}$
Matthew likes the Med	
Katie likes the Med and Rio	
Franco likes Frasca	
Caroline likes the Med and Rio	
Serves	$Serves = \{\langle e,j\rangle, \langle f,i\rangle, \langle e,h\rangle\}$
Med serves eclectic	
Rio serves Mexican	
Frasca serves Italian	

Figure 17.2 A model of the restaurant world.

17.3 Firts-Order Logic (FOL)

First-Order Logic (FOL) is flexible approach to representing knowledge that satisfies many of the desiderata for meaning representation. **First order logic (FOL) is a mathematical thing.**

Basic elements of FOL:

$$
\begin{array}{rcl}
\textit{Formula} & \rightarrow & \textit{AtomicFormula} \\
& | & \textit{Formula Connective Formula} \\
& | & \textit{Quantifier Variable}, \ldots \textit{Formula} \\
& | & \neg \textit{Formula} \\
& | & (\textit{Formula}) \\
\textit{AtomicFormula} & \rightarrow & \textit{Predicate}(\textit{Term}, \ldots) \\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term}, \ldots) \\
& | & \textit{Constant} \\
& | & \textit{Variable} \\
\textit{Connective} & \rightarrow & \wedge \mid \vee \mid \Rightarrow \\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & A \mid \textit{VegetarianFood} \mid \textit{Maharani} \cdots \\
\textit{Variable} & \rightarrow & x \mid y \mid \cdots \\
\textit{Predicate} & \rightarrow & \textit{Serves} \mid \textit{Near} \mid \cdots \\
\textit{Function} & \rightarrow & \textit{LocationOf} \mid \textit{CuisineOf} \mid \cdots
\end{array}
$$

Example: *Ay Caramba is near ICSI*
Near (LocationOf (AyCaramba), LocationOf (ICSI))

Constants: *AY Caramba, ICSI*
Specific objects in the world.
Usually in capital letters or single capitalized words resembling proper nouns. Eg. *VegetarianFood*

Functions: *LocationOf (.)*
Objects without explicit naming
Implicit objects
Functions do not return T or F, they return objects.

Predicates: *Near(. , .)*
Returns true or false (and not an object)

- Constants
  - A, VegetarianFood
- Variables
  - x, y, z
- Predicates
  - Serves, Near
- Functions
  - LocationOf, CuisineOf

- Connectives
  - $\wedge$ (and)
  - $\vee$ (or)
  - $\rightarrow$ (implications)
- Quantifiers
  - $\forall$ (all)
  - $\exists$ (there exists)

Maharani is a restaurant
○ Restaurant (Maharani)

Maharani serves vegetarian food
○ Serves (Maharani, VegetarianFood)

There is a restaurant that serves vegetarian food
○ $\exists x$ restaurant(x) $\wedge$ Serves (x, VegetarianFood)
$\exists x$ → Existencial qualifier (there exist an x). There exist an object x for which the formula holds.

Maharani is not a restaurant
○ $\neg$ Restaurant (Maharani)

All vegetarian restaurants serve vegetarian food

○ ∀x VegetarianRestaurant (x) → Serves (x, VegetarianFood)
X → variable. It refers to objects without naming specific one
∀x → universal qualifier (for all x). For all possible vales of x, the formula holds.

If x is a vegetarian restaurant, it also serves vegetarian food.
The implication symbol? Why is it there? is an "if statements". For all x if it is a vegetarian restaurant, then it serves v food. For all the objects in the world, if they serve vegetarian food, then, they are vegetarian restaurant. If I had an "and" instead of the —>, for example, if I take a table, this will still be true. It depends on the objects of the world. If you already know that all the objects that you have are vegetarian, them you do not need this implication statement.

The restaurant Ay Caramba serves Mexican food
○ Restaurant (AyCaramba) ∧ Serves (AyCaramba, MexicanFood)

Logical Connectives and truth table
I want to be able to combine the parts to make them more complex with **Logical Connectives:**

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$
False	False	True	False	False	True
False	True	True	False	True	True
True	False	False	False	True	False
True	True	False	True	True	True

Not And Or Implication, Causation ← Mirar!

Lambda Notation

- λx.P(x)
- λx.P(x)(A)
- P(A)

Used to eliminate variables in the formula.
By filling out the variable in the formula, you use **λ-reduction** to specify the variable.

Consider the following:

- λx.λy.Near(x,y)
- λx.λy.Near(x,y)(Baraco)
- λy.Near(Baraco,y)

By specifying x in the formula, we can say Baraco is near y

- λy.Near(Baraco,y)(Centro)
- Near(Baraco,Centro)

Inference:

Inference, or deduction is the most important desiderata. Ability to add valid new prepositions to knowledge base or to determine the truth of prepositions not explicitly contained within a knowledge base.

Modus ponens: the most widely used inference method. It corresponds to the *if-then* reasoning. The formula below the line can be interfered from the formulas above the line.

$$\frac{VegetarianRestaurant(Leaf) \qquad \forall x VegetarianRestaurant(x) \Rightarrow Serves(x, VegetarianFood)}{Serves(Leaf, VegetarianFood)}$$

Inference

- Given that
  - *VegetarianRestaurant*(*Rudys*)
  - $\forall x$: *VegetarianRestaurant*(x) $\Rightarrow$ *Serves*(x, *VegetarianFood*)
- What can we say about
  - *Serves*(*Rudys*, *VegetarianFood*)

Inference

- Given that
  - *Serves*(*Rudys*, *VegetarianFood*)
  - $\forall x$: *VegetarianRestaurant*(x) $\Rightarrow$ *Serves*(x, *VegetarianFood*)
- What can we say about
  - *VegetarianRestaurant*(*Rudys*)

Modus ponens

$$\frac{\alpha \qquad \alpha \Rightarrow \beta}{\beta}$$

Modus tollens

$$\frac{\alpha \Rightarrow \beta \qquad \neg\beta}{\neg\alpha}$$

α
α → β

β

VegetarianRestaurant(Leaf)
∀xVegetarianRestaurant(x) → Serves(x,VegetarianFood)

Serves(Leaf, VegetarianFood)

–

$\forall x Human(x) \rightarrow Mortal(x)$
$Human(Socrates)$

Therefore:

$Mortal(Socrates)$

Two ways of applying modus ponens:

Forward chaining: (mirar)

- Using all applicable rules to come to a conclusion
- The previously shown inferences are forward chaining
- Advantages:
- Disadvantages:

Backward chaining: (mirar)

- Using Queries to determine whether some formula is true by determining if it is present in the knowledge base. If it is not, then the next step is to search for applicable implication rules present in the knowledge base.
- ProLog (a logic programming language)

Neither Forward or backward is complete. This means that there are valid inferences that cannot be found by systems using these methods alone. Another technique, called **Resolution**, is complete. But it is computationally very expensive.

An example of backward chaining:

Does Leaf serve vegetarian food?
- ➢ Serves(Leaf,VegetaianFood)?
- ➢ ∀x vegetarianRestaurant(x) → Serves(x, VegetarianFood)
- ➢ VegetarianRestaurant(Leaf)

Neither backward chaining nor forward chaining are complete methods, they do not capture all valid inferences. However, these methods are computationally cheap, making them useable

17.4 Event and State Representation

Linguistically relevant concepts:
1. Categories
2. Events
3. Time

1. Categories:

Maharani is a vegetarian restaurant
- ISA(Maharani;VegetarianRestaurant)
- ISA → is a

Vegetarian restaurants are a sub-category of restaurants
- AKO(VegetarianRestaurant; Restaurant)
- AKO → a kind of

2. Events:

- *I ate*
- *I ate a turkey sandwich*
- *I ate a turkey sandwich at my desk*
- *I ate at my desk*
- *I ate lunch*
- *I ate a turkey sandwich for lunch*
- *I ate a turkey sandwich for lunch at my desk*

- $Eating_1(Speaker)$
- $Eating_2(Speaker, TurkeySandwich)$
- $Eating_3(Speaker, TurkeySandwich, Desk)$
- $Eating_4(Speaker, Desk)$
- $Eating_5(Speaker, Lunch)$
- $Eating_6(Speaker, TurkeySandwich, Lunch)$
- $Eating_7(Speaker, TurkeySandwich, Lunch, Desk)$
- Different kinds of eating

Different kinds of eating are not the same. While we would like to think that all these examples denote the same kind of event, predicates in FOL have a fixed **arity** – they take a fixed number of arguments. The solution at the right is highly costly and not the most efficient.

We have many different types of eating here, they describe diff things but almost in the same way. 7 gives an extra info than 6. Eating 6 follows eating 7. If we know 7 we also know 6. So, a way to write this down is the **events. —> Meaning postulates.** If there is a relationship like the one in 6 and 7 we can rewrite the rule with events.

The A (reverse) stands all
The E (reverse) stands for exist

Solution: **Meaning postulates** $\forall w, x, y, z : Eating_7(w, x, y, z) \Rightarrow Eating_6(w, x, y)$
This approach directly yields the obvious logical connection among these formulas.
However, the solution does not scale well and have some PROBLEMS:

- It makes too many commitments. The presence of this argument implicitly makes it the case that all eating events are associated with a meal (breakfast, lunch, dinner...). Ex: *for lunch*

  ∃ w,xEating(Speaker,w,x,Desk)
  ∃ w,xEating(Speaker,w,Lunch,x)
  ∃ w,xEating(Speaker,w,Lunch,Desk)

- If we knew that the first 2 formulas refer to the same event(e) we could combine them and cerate the third one. However, we have no way to tell that. SOLUTION: **Event variable** (reification – create an event object to make assertions about the same event)

  ∃ e Eating(e, Speaker,Sandwich,Lunch,Desk) ∧ Time(e, Tuesday)

Now, we can **eliminate the dichotomy** (fixed numbers of arguments) by capturing all the event arguments with additional relations, adding new predicates and summing them together. – This is called neo-Davidsonian event representation.

$$\exists e : \\ Eating(e) \wedge Eater(e, Speaker) \wedge Eaten(e, TurkeySandwich) \wedge \\ Meal(e, Lunch) \wedge Location(e, Desk) \wedge Time(e, Tuesday)$$

3. Time:

I arrived in New York

$$\exists e : \\ Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NewYork)$$

This encodes the event, but ignores time information

Solution:

- *I arrived in New York*
- $\exists e, i, p : \\ Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NewYork) \wedge \\ IntervalOf(e, i) \wedge EndPoint(i, p) \wedge Precedes(p, Now)$

- *I am arriving in New York*
- $\exists e, i : Arriving(e) \wedge Arriver(e, Speaker) \wedge \\ Destination(e, NewYork) \wedge IntervalOf(e, i) \wedge Member(i, Now)$

- *I will arrive in New York*
- $\exists e, i, p : \\ Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NewYork) \wedge \\ IntervalOf(e, i) \wedge EndPoint(i, p) \wedge Precedes(Now, p)$

The two-place predicate *Precide* represents the notion that the first time-pint argument precedes the second time. The constant *Now* refers to the current time.

However, it is more complicated than that. We need the notion of **reference point i**ntroduced by **Reichenbach.**

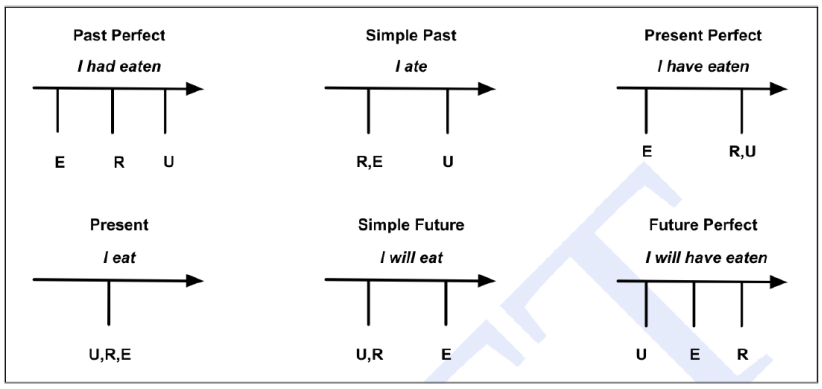


Figure 3: Fig. 3: Reichenbach's approach applied to various English tenses.In these diagrams,time flows from left to right, an **E** denotes the time of the event, an **R** denotes the reference time,and an **U** denotes the time of the utterance.

17.5 Description Logic – FER p 606

Problems

Uncertainty: Perhaps something is true

In first order logic you cannot express uncertainty (ex: I believe, I think.)… You need more complex representations. But the more complex or expressive we get, harder for the computer. The simpler the better for the computational power but we also need more expressive and more complex to represent all we want to be represented. There is always this trade off.

Believe: Reasoning over what people believe

Questions: Not statements (true/false)

...

Exercises: Do the FOL translation of ... (solutions in the PPT)

- Vegetarians do not eat meat

- Not all vegetarians eat eggs

- Some vegetarians only eat eggs

- There is exactly one vegetarian who eats meat

Chapter 18. **Computational Semantics**

20/09/2018

Semantic analysis: (last chapter) compose, assign meaning representations to linguistic input. Its main difficulties are ambiguity (as usual), and non-literal meaning (ex: idioms). Last chapiter was about how to represent meaning.

Syntax-driven semantic analysis: meaning of sentence composed from meaning of parts. Takins into account:

- Semantic of words
- Syntactic relations between words.

But, only extracts literal meaning of sentence. Literal representation may serve as input for further processing

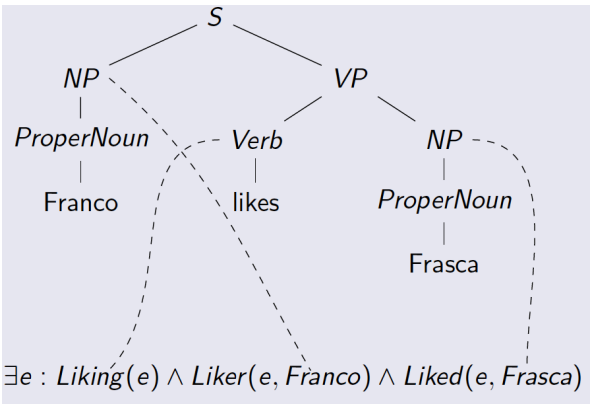
18.1 Syntax-Driven Semantic Analysis

It is the approach of **principle of compositionality**. The meaning of the sentence can be constructed from the meaning of the parts.

In syntax-driven semantic analysis, the composition of meaning representations is guided by the **syntactic components** and **relations provided** by the grammars discussed in Chap.12. (Contex-Free Grammar).

Ambiguity in semantic analysis → syntactic parser may produce several analyses for a sentence (due to the structural ambiguity). ALSO, words in a sentence may have ambiguous meaning (even if the syntactic category is unambiguous). → both deal to ambiguous meaning representations.

Syntactic-driven analysis example:



18.2 Semantic Augmentation to Syntactic Rules

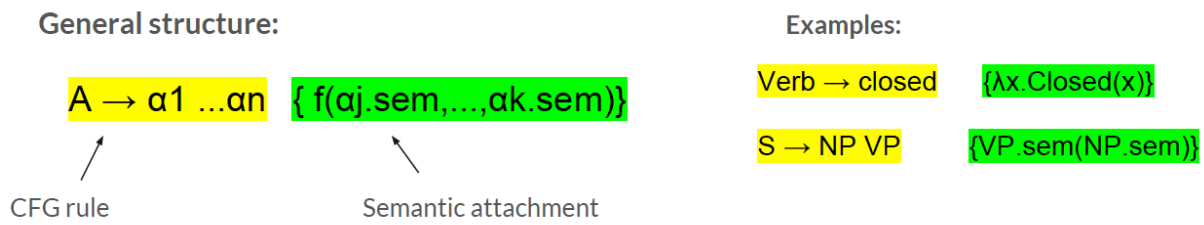
We attach the semantic attachment right after the Context Free Grammar (CFG)

Augment our context-free grammar rules with **semantic attachments**. (extend CFG's with descriptions that provides us with instructions that specify how to compute the meaning representation of a construction from the meaning of its constituent parts.).

$$A \rightarrow \alpha_1 \ldots \alpha_n \quad \{f(\alpha_j.sem, \ldots, \alpha_k.sem)\}$$

The formula means that the representation assigned to the construction of A (named *A.sem*) can be computed by running the function *f* on some subset of the semantic attachments of As constituents.

We build the meaning representation up as we parse the data, putting them together according to the rules.



A refresher: the Lambda notation and reduction

The **λ** is used to indicate that the variable (the letter) following it needs to be filled in.

If a logic expression starts with **λ.x** it means that the **x** in the following formula is a placeholder. It defines a "template" with parameter x.

From **λ.x.closed(x)** to **λ.x.closed(x) (A)** to **closed(A)**

Scope of a variable: from the moment we introduce it until the end of the formula.

Each context FG rule has been complemented with a semantic attachment. We start filling the gaps in the order of the lambdas in the left (from left to right). They are the ones indicating the order. You have to keep going until you do not find any more lambda. - **it is called lambda reduction. (it has to do with the scope).**

The lambda introduces the variable into the scope. We do not know what the variable will be substituted for yet, but it is important to introduce the variable into the scope first.

Sentence: Maharani closed

Building semantic representation:

1. VP.sem(NP.sem)
2. λx.closed(x) (Maharani)
3. closed(Maharani)

Rules:

S → NP VP	{VP.sem(NP.sem)}
NP → ProperNoun	{ProperNoun.sem} = {Maharani}
VP → Verb	{Verb.sem} = {{λx.Closed(x)}
ProperNoun → Maharani	{Maharani}
Verb → closed	{{λx.closed(x)}

The book does this bottom-up.

Another example:

Sentence: Every restaurant closed

Rules:

S → NP VP	{VP.sem(NP.sem)}
NP → Det Nominal	{Det.sem(nominal.sem)}
VP → Verb	{Verb.sem}
Nominal → Noun	{Noun.sem}
Verb → closed	{λz. ∃ eClosed(e) ∧ Closed(e,z)}
Noun → restaurant	{λ.x.Restaurant(y)}
Det → every	{λP.λQ. ∀ xP(x) ⇒ Q(x)}

What the… is this monstrosity?

Intermezzo: Restriction

$$\{\lambda P.\lambda Q.\forall x P(x) \Rightarrow Q(x)\}$$

- Semantic attachment of the world 'every'
- The variables P and Q used here need to be filled in at a later date (indicated by λ)
  - Where P is a noun and Q is verb.
- For every x
- Where X is P
- X does Q

- We have to notate it this way, because only stating **λP.∀xP(x)** we would mean something like 'for every x, x is a p'. (for example: "everything is a restaurant").
- We need to add the implication to show we are saying something about *ALL P's*.
- Since we don't know what we are saying about them (what they are doing) we use placeholder Q.

Back to business

Sentence: Every restaurant closed

Let's start by figuring out NP.sem:

1. Det.sem(nominal.sem)
2. λP.λQ.∀xP(x) ⇒ Q(x)(λ.y.Restaurant(y))
3. λQ.∀xλ.y.Restaurant(y)(x) ⇒ Q(x) We first replace the leftmost lambda-term: the P.
4. λQ.∀xRestaurant(x) ⇒ Q(x) By embedding, we can solve another lambda-expression.

We see now that the NP.sem is **λQ.∀xRestaurant(x) ⇒ Q(x)** (read: for every x where x is a restaurant, x does Q)

Rules:

S → NP VP	{VP.sem(NP.sem)}
NP → Det Nominal	{Det.sem(nominal.sem)}
VP → Verb	{Verb.sem}
Nominal → Noun	{Noun.sem}
Verb → closed	{λz. ∃eClosed(e) ∧ Closed(e,z)}
Noun → restaurant	{λ.y.Restaurant(y)}
Det → every	{λP.λQ.∀xP(x) ⇒ Q(x)}

Continuing...

Sentence: Every restaurant closed
NP.sem = λQ. ∀ xRestaurant(x) ⇒ Q(x)
VP.sem = λx. ∃ eClosed(e) ∧ Closed(e,x)
S.sem (our target!) = NP.sem(VP.sem)

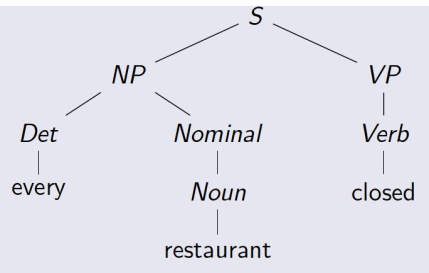
Rules:

S → NP VP	{NP.sem(VP.sem)}
NP → Det Nominal	{Det.sem(nominal.sem)}
VP → Verb	{Verb.sem}
Nominal → Noun	{Noun.sem}
Verb → closed	{λz. ∃ eClosed(e) ∧ Closed(e,z)}
Noun → restaurant	{λ.y.Restaurant(y)}
Det → every	{λP.λQ. ∀ xP(x) ⇒ Q(x)}

Let's plug it all in our S.sem formula!

1. NP.sem(VP.sem)
2. λQ. ∀ xRestaurant(x) ⇒ Q(x) (λz. ∃ eClosed(e) ∧ Closed(e,z))
3. ∀ xRestaurant(x) ⇒ λz. ∃ eClosed(e) ∧ Closed(e,z)(x) Replacing the leftmost lambda term (Q)
4. ∀ xRestaurant(x) ⇒ ∃ eClosed(e) ∧ Closed(e,x) And again. This time we replace the leftmost lambda (y)
5. ∀ xRestaurant(x) ⇒ (∃ eClosed(e) ∧ Closed(e,x)) **Final representation!**

This says something like "For every x where x is a restaurant, there was an event (closed) denoted by e, that applied to it." or, in normal human terms: **every restaurant closed.**



18.3 Quantifier Scoping Ambiguity and Underspecification

- Every restaurant has a menu
- $\forall x : Restaurant(x) \Rightarrow \exists y : Menu(y) \wedge \exists e : Having(e) \wedge Haver(e, x) \wedge Had(e, y)$
- All restaurants have menus
- $\exists y : Menu(y) \wedge \forall x : Restaurant(x) \Rightarrow \exists e : Having(e) \wedge Haver(e, x) \wedge Had(e, y)$
- There is a menu that all restaurants share

Expressions containing qualified terms can give rise to ambiguous representations.

Quantifier scoping ambiguity: The difference between the two interpretations given above arises from which of the two qualified variables has the outer scope.

The approach outlined in the last section cannot handle this phenomenon. The interpretation that is produced is based on the order in which the lambda expressions are reduced as dictated by

the grammar and its semantic attachments. This problem can't be handled by beta-reduction, as the order in which lambda-expressions are reduced is dictated by the grammar and it's syntactic attachments.

To fix this problem, we need:

- The ability to efficiently create *underspecified representations* that embody all possible readings without explicitly enumerating them.
- A means to generate, all of the possible readings from this representation
- The ability to choose among the possible readings. [NO SOLUTON, YET]

18.3.1 STORE AND RETRIEVE APPROCHES

Underspecified representation – Cooper storage

A way to address the qualifier scoping ambiguity.
We introduce the notion of **Cooper storage**: In our original approach to semantic analysis, we assigned a single FOL formula to each node in a parse tree. In the new approach, we replace this single semantic attachment with a store.
This store includes a core meaning representation for a node in a parse tree, along with an indexed list of quantified expressions gathered from the nodes below this node in the parse tree. When needed, you can retrieve the scope quantifiers at a later point from storage.

A store would look like this:

$\exists e\ Having(e) \wedge Haver(e,s1) \wedge Had(e,s2)$
$(\lambda Q.\ \forall x Restaurant(x) \Rightarrow Q(x),1),$
$(\lambda Q.\ \exists x menu(x) \wedge Q(x),2)$

When you pull out the second element of the store first:

$\lambda Q.\ \exists x(menu(x) \wedge Q(x))$
$(\lambda s2.\ \exists e\ Having(e) \wedge Haver(e,s1) \wedge Had(e,s2)$

It reduces to:

$\exists e\ (Menu(x) \wedge \exists e\ Having(e) \wedge Haver(e,s1) \wedge Had(e,x))$

Then, when you pull the first element out of storage after that:
$\lambda Q.\ \forall x(Restaurant(x) \Rightarrow Q(x))$
$(\lambda s1\ \exists e\ (Menu(x) \wedge \exists e\ Having(e) \wedge Haver(e,s1) \wedge Had(e,x)))$

You can reduce this to:

$\forall x\ Restaurant(x) \Rightarrow \exists y(Menu(y) \wedge \exists e\ (Having(e) \wedge Haver(e,x) \wedge Had(e,y)))$

Cooper storage:

- Do not specify order of quantifiers
- Keep expressions with quantifiers in a set
- Keep track of where expressions go
- Apply expressions in certain order to generate meaning

So, what do you analyze first? *This is what Cooper tries to solve in his store and retrieve approaches.* It depends on the quantifier that you select first. Ex: There is one menu for all the restaurants (menu first) // Here all the restaurants, they have a menu (restaurant first)

An underspecified representation of a sentence should specify what we know about how these representations combine and no more. It should also remain agnostic about the placement of the quantifiers in the final representation, we introduce the notion of **Cooper storage** and once again leverage the power of λ-expressions. Instead of assigning a single FOL formula to each node in a parse tree, the new approach replaces the single semantic attachment with a store. The store includes the core meaning representation for a node along with an indexed list of quantified expressions gathered from the nodes below this node in the tree. These expressions are in the form of λ-expressions.

Constraint-Based Approaches – Hole Semantics

Constraint-based approaches:

- Address wider range of ambiguities (e.g. negation)
- Hole semantics
  - Provide partial ordering on applications of expressions
  - Apply expressions based on partial ordering.

Lambda-variables are being replaced with ¨Holes¨, and instead of filling these out with lambda-reductions, we first add labels to all candidate First-Order Logic subexpressions.

In order to prevent us from filling the holes at random, dominance restraints were added to restrict which labels can fill which holes. This way we can make additional rules to fill the gaps, an ability our previous approach hadn't.

But there are things that Cooper Storage cannot solve either. Ex: ((Old men) and women) <— do not understand that. IMPORTANT! Demanar apunts.

Unfortunately, the storage-based approach defined earlier suffers from two problems:

- First, **it only addresses the problem of scope ambiguities introduced by quantified noun phrases**.
- Second, even though it allows us to enumerate all the possible scoping's for a given expression, it **doesn't allow us to impose additional constraints on those possibilities**. While this is crucial in applying specific lexical, syntactical, and pragmatic knowledge to narrow down the range of possibilities for any given expression.

A fix for the above problem would be to use the **hole semantics** approach. The Q predicates in the quantified expressions are place-holders that will eventually be replaced by arbitrary FOL expressions through a series of coordinated λ-reductions. In the hole semantics approach, we replace these λ-variables with holes instead of using λ-reductions. In a fully-specified formula, all holes will be filled with labelled subexpressions. To avoid random filling, we'll add **dominance constraints** between holes and labels that restrict which labels can fill which holes.

(18.25) Every restaurant has a menu.

$$l_1: \forall x Restaurant(x) \Rightarrow h1$$
$$l_2: \exists x Menu(y) \wedge h2$$
$$l_3: \exists e\ Having(e) \wedge Haver(e, x) \wedge Had(e, y)$$
$$l_1: \leq h_0, l_2 \leq h_0, l_3 \leq h_1, l_3 \leq h_2$$

This constraint-based approach to underspecification addresses many of the problems with the store-and-retrieve approach that we raised at the beginning of this section. First, the approach is not specific to any particular grammatical construction or source of scope ambiguity. Second, dominance constraints give us the power to express constraints that can rule out unwanted interpretations.

18.5 Integration of Semantics into the Early Parser

It is possible to perform semantic analysis in parallel with syntactic processing.

To integrate semantic analysis into an Earley parser you need to modify the original algorithm:

1. The rules of the grammar are given a new field to contain their semantic attachments
2. The states in the chart are given a new field to hold the meaning representation of the constituent
3. The ENQUEUE function is altered so that when a complete state is entered into the chart, its semantics are computed and stored in the state's semantic field.

Advantage integrated approach: The APPLY-SEMANTICS can fail in a similar way that unification can fail.

Disadvantage integrated approach: Too much effort may be spent on the semantic analysis of orphan constituents that do not in the end contribute to a successful parse.

18.6 Idioms and Compositionality

The **principle of compositionality** runs into trouble fairly quickly when real language is examined. This is because there are many cases in which the meaning of a constituent is not based on the meaning of its parts.
The most straightforward way to handle idiomatic constructions is to **introduce new grammar rules** specifically designed to handle them. These idiomatic rules mix lexical items with grammatical constituents and introduce semantic content that is not derived from any of its parts.

Ex: Thank you, Good morning —> these kinds of situations you can just put them in the idiom because they do not accept any kind of flexibility. You cannot put any word in between.

It is not easy to deal with the idioms. And this is an option you have to make when building your lexicon.

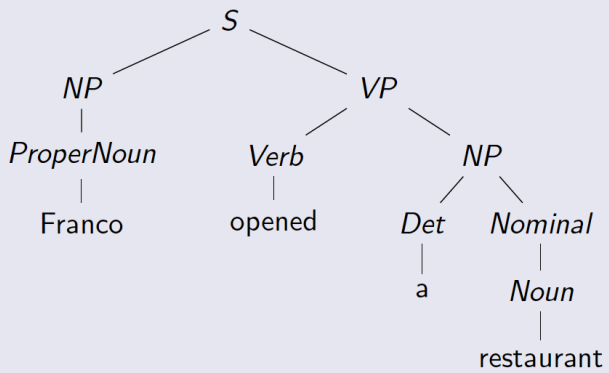
Sentence-level attachments

- Declarative sentences
  - $S \rightarrow NP\ VP\ \{DCL(NP.sem(VP.sem))\}$
- Imperative sentences
  - $S \rightarrow VP\ \{IMP(VP.sem(DummyYou))\}$
- Yes-no questions
  - $S \rightarrow Aux\ NP\ VP\ \{YNQ(VP.sem(NP.sem))\}$
- Sentence-level predicates may be interpreted by system

Semantics exercice

Franco opened a restaurant

Exercise: parsing



Exercise: Semantics

- ProperNoun(Franco): $\{\lambda f.f(Franco)\}$
- NP(ProperNoun((Franco))): $\{ProperNoun.sem\}$
- NP(ProperNoun((Franco))): $\{\lambda f.f(Franco)\}$
- Verb(opened): $\{\lambda w.\lambda z.w(\lambda y.\exists e : Opening(e) \wedge Opener(e,z) \wedge Opened(e,y))\}$
- Det(a): $\{\lambda P.\lambda Q.\exists x : P(x) \wedge Q(x)\}$
- Noun(restaurant): $\{\lambda r.Restaurant(r)\}$
- Nominal(Noun(restaurant)): $\{Noun.sem\}$
- Nominal(Noun(restaurant)): $\{\lambda r.Restaurant(r)\}$

Exercise: Semantics

- NP(Det(a),Nominal(Noun(restaurant))): $\{Det.sem(Nominal.sem)\}$
- NP(Det(a),Nominal(Noun(restaurant))): $\{(\lambda P.\lambda Q.\exists x : P(x) \wedge Q(x))(\lambda r.Restaurant(r))\}$
- NP(Det(a),Nominal(Noun(restaurant))): $\{\lambda Q.\exists x : (\lambda r.Restaurant(r))(x) \wedge Q(x)\}$
- NP(Det(a),Nominal(Noun(restaurant))): $\{\lambda Q.\exists x : Restaurant(x) \wedge Q(x)\}$

Exercise: Semantics

- VP(Verb, NP): $\{Verb.sem(NP.sem)\}$
- VP(Verb, NP): $\{(\lambda w.\lambda z.w(\lambda y.\exists e : Opening(e) \wedge Opener(e,z) \wedge Opened(e,y)))(\lambda Q.\exists x : Restaurant(x) \wedge Q(x))\}$
- VP(Verb, NP): $\{\lambda z.(\lambda Q.\exists x : Restaurant(x) \wedge Q(x))(\lambda y.\exists e : Opening(e) \wedge Opener(e,z) \wedge Opened(e,y))\}$
- VP(Verb, NP): $\{\lambda z.(\exists x : Restaurant(x) \wedge (\lambda y.\exists e : Opening(e) \wedge Opener(e,z) \wedge Opened(e,y))(x))\}$
- VP(Verb, NP): $\{\lambda z.\exists x : Restaurant(x) \wedge \exists e : Opening(e) \wedge Opener(e,z) \wedge Opened(e,x)\}$

Exercise: Semantics

- S(NP, VP): $\{NP.sem(VP.sem)\}$
- S(NP, VP): $\{(\lambda f.f(Franco))(\lambda z.\exists x : Restaurant(x) \wedge \exists e : Opening(e) \wedge Opener(e,z) \wedge Opened(e,x))\}$
- S(NP, VP): $\{(\lambda z.\exists x : Restaurant(x) \wedge \exists e : Opening(e) \wedge Opener(e,z) \wedge Opened(e,x))(Franco)\}$
- S(NP, VP): $\{\exists x : Restaurant(x) \wedge \exists e : Opening(e) \wedge Opener(e,Franco) \wedge Opened(e,x)\}$

There are 2 approaches to tackle this problem:

- **The Store and Retrieve Approaches:** This is really diff to understand. Teacher does not expect us to understand everything.

WE DO NEED TO KNOW that there are **2 types of ambiguity:**

> **Syntactic ambiguity**: diff tree structures. Because of the diff tree structures you get a diff meaning. Ex. Elephant in the pajamas
>
> **No syntactic ambiguity but diff meanings: T**here is only one syntactic analysis but still off meanings: Farmes bitting a donkey

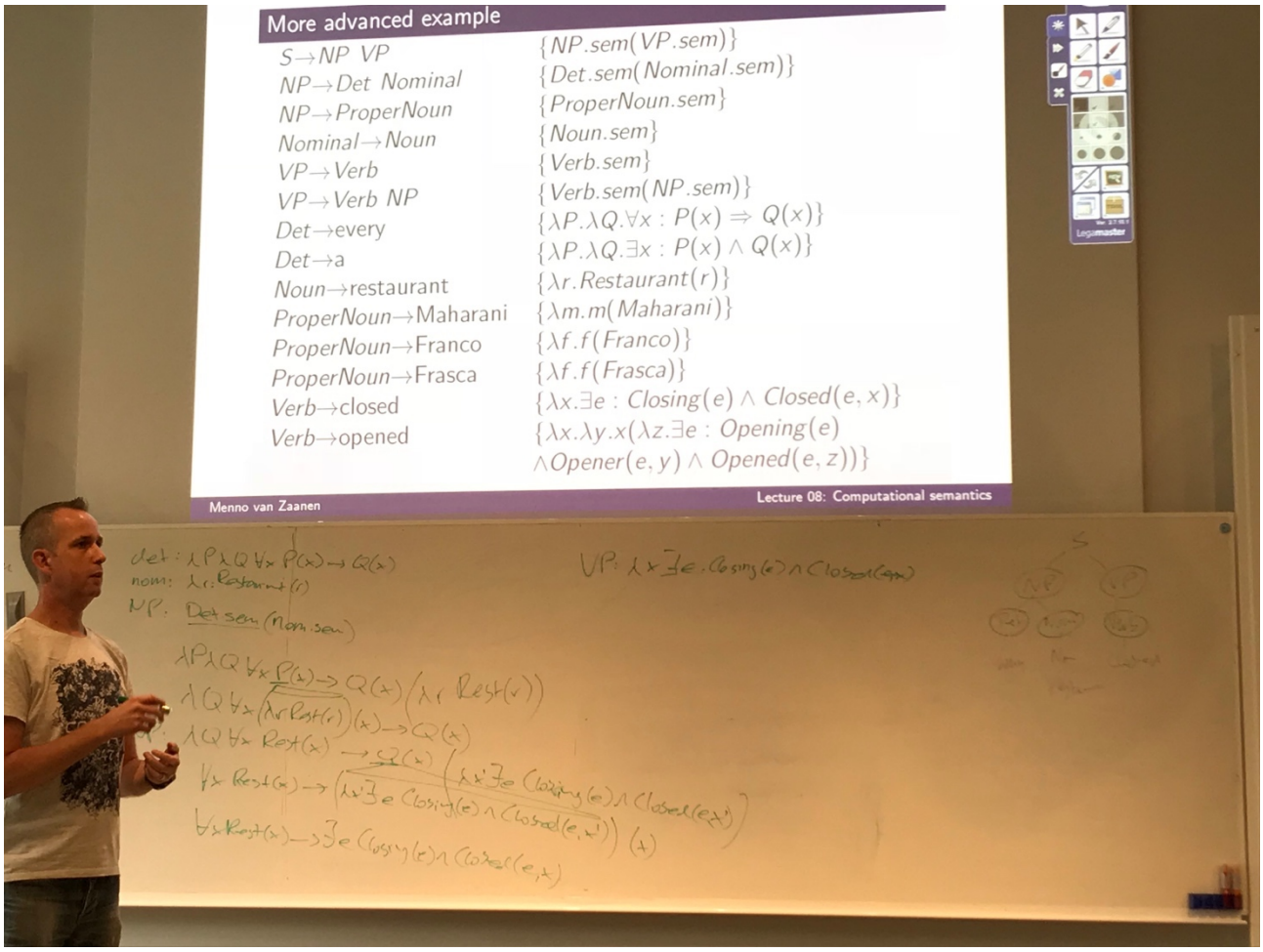
Example exam question: What do you use Cooper Store for?

How do we integrate the semantics into parsing?

First thing you do is start parsing the sentence with CFG. Now, we want to describe the meaning. First parsing and the applying the semantics (meaning) with the semantic attachments.

PHOTO - How to do it.

Every restaurant closed



( ) determins the scope. Outside the ( ) (ex of the “r” in the photo) the variable does not exist anymore.

Chapter 19. **Lexical Semantics –** *Apunts companys*

24/09/2018

Lexical semantics is a linguistic field that is concerned with the study of word meaning. Meaning(s) of words and the relations between meaning. A word can have several meanings, each one is called ***word senses.*** Meaning typically depends on context.

Resources: WordNet, PropBank, FrameNet

A word is a **lexeme** (when you do not know what the word will be yet).

Tokenization: split text into tokens (often words). Non-trivial (especially in some languages, like Chinese)
Segmentation: Split text into segments, either sentences or phrases

Grouping related words

Stemming: Reduce word to its stem
Lemmatization: Reduce word to its lemma
Aim: reduce inflectional forms (Stemming removes affixes, Lemmatization maps to “dictionary entry”)

19.2 Relations between words/senses

Homonymy

Polysemy

Synonymy

Antonymy

Metonymy

Pars pro toto

Totum pro parte

Hyponymy

Hypernymy

Meronymy

Holonymy

Homophony

Homography

19.4 Event participants

Describe meaning of events. Given an event, describe relationships of participants.
Approaches:

- Semantic role
- Selectional restrictions

Example

- Sasha broke the window
  - Breaking event
  - Sasha is the breaker
  - The window is what's being broken
- Pat opened the door
  - Opening event
  - Pat is the opener
  - The door is what's being opened

Instead of specific names they are called arguments *agent*
Also, window and door are called *theme*

Other commonly used relations:

Agent	volitional causer of event
Experiencer	experiences of event
Force	non-volitional causer of event
Theme	participant most directly affected by event
Result	end product of event
Content	proposition or content of propositional event
Instrument	instrument used in event
Beneficiary	beneficiary of event
Source	origin of object of transfer event
Goal	destination of object of transfer event

Usefulness of semantic relations:

Syntax does not correspond well to semantic structure.
Semantic relations provide shallow semantics.

Example: - They all have the same meaning but different pragmatics
John gives Mary a book
Mary is given a book by John
A book is given to Mary by John

Thematic roles are used for:

- Question answering
- Machine Translation
- Information extraction

Granularity of relations: Thematic roles are too coarse-grained. Ex:
The cook opened the jar with the gadget
The gadget opened the jar
She ate the banana with a fork
**A fork ate the banana*

Selectional Restrictions

Define restrictions on arguments of verbs. Ex. Eat requires argument that is edible
This can help to disambiguate analyzes of sentences

Primitive decomposition

Try to Split meaning of Word into smaller parts of meaning
Hopefully using only small set of primitive meaning parts

Hard to identify small set of primitive meaning parts

Important things to understand about the chapter:

The chapter describes a lot of info about how to give meaning to a word

There are 2 views:

- **Linguistics** are interested in this one: I want to be able to describe the difference. Why sometimes reversing the argument works and why sometimes it does not work? What is possible for some sentences why it is not possible for others? - Then you get all the semantic constrains. IT IS ABOUT UNDERSTANDING HOW LANGUAGE WORKS. Which things are valid in a language. Not much to do with computation. Semantics. Some sentence

- **Computational** point of view. Be able to extract info from a sentence. You do not need to describe in so much detail. Syntaxs helps in some way to understand the meaning of the sentence but there is not always a perfect match between syntactics and semantics, that is why it is really hard.

WordNet: data set that you can use for all kinds of stuff. WordNet for English is really big. People is trying to assign id numbers to the English words to their own language words, so it works like a kind of a translation.

FrameNet:

Questions about Edit Distances - assignment 1

Find a way convert one word to another word.

- Insert a letter - cost: 1
- Delete a letter - cost: 1
- Subsitute a letter for another letter - cost: 2
- (Anotehr one)

Each operations have particular cost.

First, you need to calculate how expensive it is, and you do so with the Matrix (in the document).

Ex. How **abcf** can be converter to **dbe —- apunts a la llibreta**

Chapter 20. **Computational Lexical Semantics**

27/09/2018

Computational Lexical Semantics: computing with words meanings. Important role of context and similarity of senses. Goal is to resolve or reduce ambiguity at the word level.

Word sense disambiguation (WDS): task of examining word tokens in context and determining which sense of each word is being used. This will lead us to **word semantic similarity**.
Automatically select correct sense of a word.
Input: word in context, fixed) inventory of senses.
Output: correct sense for word in context.

20.1 Word Sense Disambiguation

In Chapter 18 we ignored the issue of lexical ambiguity and this is unreasonable. Without selecting the correct sense for the words in an input, the enormous amount of homonymy and polysemy in the lexicon would quickly overwhelm any approach.

Applications of WSD: - in each app we deal with WSD differently. Here we present it as stand-alone task. Also, the data set (input) must depend on the task.

- Machine Translation
- Question answering
- Information retrieval
- Text classification
- Summarization and generation
- Automata essay grading
- Plagiarism detection

There are two variants of the WSD task:

1. **Lexical sample task:** Small pre-selected set of target words is chosen with the inventory of their senses from some lexicon. Supervised ML approaches are used because the set of words and the set of senses are small. Classifier systems can then be trained with these labeled examples – and then, label the unlabeled with the resulting model.
   LABEL INSTANCES → TRAIN CLASSIFIER → LABEL UNSEEN INSTANCES

2. **All-word task:** Entire text and lexicon with an inventory of sense for each entry. Are required to disambiguate every content word in the text. Label all words. Similar to the PoS-tagging but with much larger tag set. There is a serious data sparseness problem: it is unlikely that adequate training data for every word in the test set will be available.

- Lexical sample:
  On this occasion she is at a great **party**/*social_party* .
- All-words:
  On this occasion/*occurrence* she is at a great/*more_than_ordinary* party/*social_party* .

We start with the (1) **supervised learning**. Then, we look at methods to deal with the lack of data for fully supervised algorithms – (2) **semi supervised Algorithms** (eg. Dictionary-based approaches and bootstrapping techniques). And finally, some notions of the (3) **unsupervised algorithms**.

Supervised learning:

- (Hand)-label data
- Extract features helpful to distinguish between senses
- Train classifier on features
- Apply classifier to unseen instances
- Evaluate classification results

Unsupervised learning:

- Automatic creation of sense inventory based on training set
- Combines clustering and distance metric between vectors
- No label for sense
- Evaluation more difficult

20.2 Supervised WSD

Hand-labeled data.
For **lexical samples** → labeled corpora for individual words
For **all-word** disambiguation → semantic concordance. A corpus in which each open-class word in each sentence is labeled with its word sense form a specific dictionary or thesaurus.

Feature Extraction for Supervised Learning:

Goal: find features to predict the word sense.
Features usually extracted from context within the sentence.

"What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?"

Approach:

1. Pre-process input (eg. Lemmatization / stemming, PoS-tagging)
2. Extract context features form enriched input
3. Store features / information in a **feature vector** for each instance.

Example for feature extraction: *Orton Chirwa formed an opposition party in exile.*

1. Pre-process sentence
   Orton/N Chirwa/N formed/V an/DET opposition/N **party/N** in/PREP exile/N ./PUNCT
2. Extract context features
   - Window of ±2 words
   - Word and its PoS
3. Feature vector for this sentence/instance
   `[an, DET, opposition, N, in, PREP, exile, N]`

Two classes of features are generally extracted from these neighboring contexts:
[non stop-words used (the, a, and, etc.)]

COLLOCATIONAL FEATURES → content of specific position in relation to target word

$$[w_{i-2}, \text{POS}_{i-2}, w_{i-1}, \text{POS}_{i-1}, w_{i+1}, \text{POS}_{i+1}, w_{i+2}, \text{POS}_{i+2}] \quad (20.2)$$

would yield the following vector:

`[guitar, NN, and, CC, player, NN, stand, VB]`

BAG-OF-WORDS → unordered set of neighboring words. Represent the context of a target word by a vector of features. Each binary feature indicating whether a vocabulary word *w* does or doesn't occur in the context.

[*fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band*]

Using these word features with a window size of 10, (20.1) would be represented by the following binary vector:

`[0,0,0,1,0,0,0,0,0,0,1,0]`

Most approaches to WSD use both collocational and bag of words, building a very long vector or by building a classifier for each feature type and combining them in some manner.

The vectors store relevant information about a data instance in machine-usable form.
Values encoding information can be **numeric** (binary (yes / no), probabilities, other measures) or **nominal** (word, PoS tag, dependency relation, combination).

Naive Bayes and Decision List Classifier:

(any supervised learning paradigm can be used to train a sense classifier, but we will focus on these two)

Choosing best sense in choosing the most probable sense.

Bayes rule: - where s_k = sense *k* of target word in context *c* (a les formules del llibre la c és f)
But, P (c) is a constant for all senses → does not influence on best class, so we can eliminate it.

$$\arg\max_{s_k} P(s_k|c) = \frac{P(c|s_k)P(s_k)}{P(c)} \qquad \arg\max_{s_k} P(s_k|c) = P(c|s_k)P(s_k)$$

Naively assume that the **features are conditionally independent** of one another given the word sense.

$$P(c|s_k) \approx \prod_{j=1}^{n} P(c_j|s_k)$$

Training this is done by estimating each of the probabilities. We get the maximum likelihood estimate (MLE) of this probability from the sense-tagged training corpus by:

$$P(s_i) = \frac{\text{count}(s_i, w_j)}{\text{count}(w_j)}$$

We also need to know each of the individual feature probabilities P (f_j I s). The MLE for these is:

$$P(f_j|s) = \frac{\text{count}(f_j, s)}{\text{count}(s)}$$

So, you take the target word in context, extract the specified features, compute for each sense (with the naive assumption formula), and return the sense associated with the highest score. Low probabilities. Collocated words not seen during the training get probability 0 → apply smoothing (e.g. add-one, Laplace).

Exercise for naive bayes

context feature	sense 1 pol. party (60)	sense 2 birthday (100)
opposition	10	2
form	20	1

- Calculate probabilty of features given a sense
  $P(opposition|pol.party) = 10/60 = 0.166$,
  $P(opposition|birthday) = 2/100 = 0.02$,
  $P(form|pol.party) = 20/60 = 0.333$,
  $P(form|birthday) = 1/100 = 0.01$
- Calculate score for
  Orton Chirwa formed an opposition **party** in exile .
  Sense_pol.party =0.166*0.333*0.375 = 0.021
  Sense_birthday =0.02*0.01*0.625 = 0.000125

→ **DECISION LIST CLASSIFIER**: Naive are diff for humans to understand. Lists classifiers are more transparent. Sequence of test applied to each feature vector.
If statements. (example in pag. 677)
If all else fails, majority sense is returned.

20.3 WSD Evaluation, Baseline, and Ceilings

Extrinsic or **task-based evaluation:**

End-to-end evaluations. – only way to test on the real application
Difficult and time consuming – they require integration of the complete system working
Only tell us about the WSD in the context of the application, results not generalizable

Intrinsic or **stand-alone evaluation** or **in vitro**:

The one normally used.
Sense accuracy → the % of words tagged as the hand-labeled in a test set (held-out)
Recall → how many out of all tagged
Precision → how many correctly tagged

Data split into **training**, (**tuning**) and **test** or **held-out** data.
We also have additional measures to allow interpretation of results:

BASELINE → tell us how well we are doing it compared to a relatively simple approach.

We normally use the most frequent sense.
And Lesk algorithm

CEALING → How close we are to the optimal performance.

Human inter-annotator. It ranges from about 75% to 80% in the WordNet, and 90% in other more coarse-grained inventories.

Although hand-labeled is still the best method for evaluation, it is expensive!
For supervised algorithms we need the labeled data anyway to train the a. Then, getting a bit more data for the evaluation seems justified.
For the **unsupervised algorithms** (20.10), we would like to have an **evaluation method that avoids labeled data:**

Pseudowords:

Banana – door
In general, better optimistic results as they are easier to disambiguate.

20.4 WSD: Dictionary and Thesaurus Methods

Hand – labeled data is expensive and time consuming → **Weakly / indirect supervised** methods for WSD are the alternative

→ Dictionary-based (e.g Lesk Algorithm)

- Based on overlapping words (non-stopwords) in a dictionary definition
  Choose the sense whose dictionary gloss or definition shares the most words with the target word's neighbors.
- Depends heavily on the length of dictionary entries. – if they are short not enough overlapping. *Solution: Corpus Lesk*
- **Corpus Lesk**: add the words of the SemCor in each entry
  - Corpus Lesk also applies a weight **(inverse document frequency, IDF)** to each overlapping word. (eg. *The, and, of...)* it uses IDF instead of the stop words list.
- We can also combine the Lesk approach with a supervised one, by adding new bag-of-words features.

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

→ Selectional restrictions (defined in chap 19)

Eat have a restriction → something must be eaten
Problem: restriction violations (eg. negations or overstated)
Modern models use it as a preference, not as a rigid requirement.

→ (20.5) Bootsrapping (the Yarowsky Algorithm is the most well known)

Start with a small set of labeled instances (seed)
Train classifier
Label unlabeled corpus
Select newly labeled examples based on confidence, add to the training corpus
Re-train classifier, etc.

One sense per collocation: certain words or phrases are always found with one sense. E.g political party vs. Birthday party

One sense per disclosure: ambiguous word appearing multiple times in a text/discourse often has same sense. Validity depends on granularity of sense inventory.

- Initial seed needs to be accurate
- Confidence measure needs to be good

→ **(20.6) Word Similarity** (thesaurus-based or distributional-based)

Computation of relations between words
Synonyms too rigid
Near-synonyms expressed as semantic distance (close = similar)

Two algorithms can be used:

Thesaurus-based: measures the distance between two senses in an on-line thesaurus like WordNet or MeSH.

Path-length based similarity: assumes uniform distance

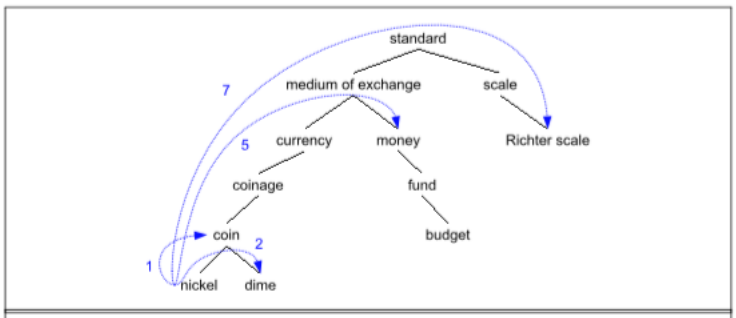


Figure 20.6 A fragment of the WordNet hypernym hierarchy, showing path lengths from *nickel* to *coin* (1), *dime* (2), *money* (5), and *Richter scale* (7).

Path-based similarity:

$$\text{sim}_{\text{path}}(c_1, c_2) = -\log \text{pathlen}(c_1, c_2)$$

The correct sense of the word similarity:

$$\text{wordsim}(w_1, w_2) = \max_{\substack{c_1 \in \text{senses}(w_1) \\ c_2 \in \text{senses}(w_2)}} \text{sim}(c_1, c_2)$$

(from all the senses of the two words, which pair of senses has the maximum sense similarity)

Information-content word similarity: add probabilistic info derived forma corpus. The P that a word is an instance of concept.

➔ Specific words have higher information content than general words
➔ Specific concepts are mentioned less often, therefore, P(c) is low
➔ Information content is represented by "-log P(c)". This makes specific words have a higher value than general words.

-log (entity) < -log (coast)

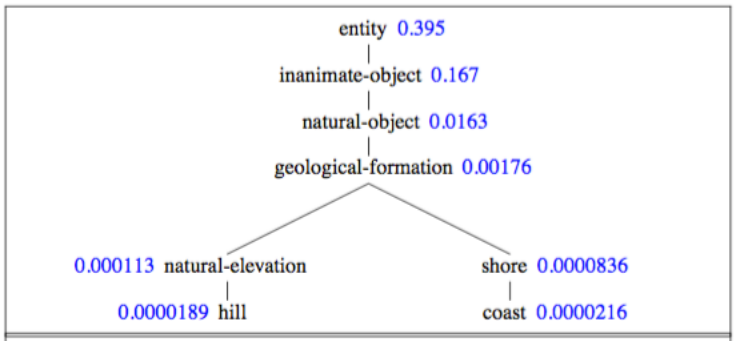


Figure 20.7 A fragment of the WordNet hierarchy, showing the probability $P(c)$ attached to each content, adapted from a figure from Lin (1998b)

Lowest common subsumer, LCS (c1, c2) = the lowest node in the hierarchy is a hypernym of both c1 and c2.

Commonality: the amount of information content shared by both words
- Look for the *P(c)* of the **lowest common subsumer (LCS)**, or the lowest hypernym that contains both of the target words.
- The LCS of 'hill' and 'coast' is 'geological formation'
- Higher commonality = higher word similarity

Difference: the amount of information needed to describe both words fully
- Use *P(c)* of the two words to calculate how different they are
- Higher difference = lower word similarity

Lin similarity: $\text{sim}_{\text{Lin}}(c_1, c_2) = \frac{2 \times \log P(LCS(c_1, c_2))}{\log P(c_1) + \log P(c_2)}$

Jiang-Conrath distance: $\text{dist}_{\text{JC}}(c_1, c_2) = 2 \times \log P(\text{LCS}(c_1, c_2)) - (\log P(c_1) + \log P(c_2))$

- **Dictionary-based method**: an extension of the Lesk algorithm
- While **thesaurus-based methods** use the hierarchy in a thesaurus to measure similarity, dictionaries use **glosses** (definitions) of the words.
- **Extended Gloss Overlap / Extended Lesk** measure: two concepts/senses are similar if their glosses contain overlapping words.
  - When comparing two words/concepts, we look at the overlap of not just the glosses of those specific concepts, but also of hypernyms, hyponyms, meronyms, and other relations of the two concepts
  - If we only looked at hyponyms, the equation would look like this:

$$\begin{aligned}\text{similarity(A,B)} = \ & \text{overlap(gloss(A), gloss(B))} \\ & + \text{overlap(gloss(hypo(A)), gloss(hypo(B)))} \\ & + \text{overlap(gloss(A), gloss(hypo(B)))} \\ & + \text{overlap(gloss(hypo(A)),gloss(B))}\end{aligned}$$

Evaluation of the method: **results:** Jiang-Conrath and Extended Lesk are the best two approaches. (more in page 690)

(20.7) Distribution-based algorithms: estimate word similarity by finding words that have similar distribution in a corpus.

20.8 Hyponymy and Other Word Relations

WordNet and MeSH also include hyponyms/hypernyms.
WordNet also includes antonymy, meronymy and other relations

But since many words are not in these resources, it is important to be able to learn new hypernym and metonym relations automatically. The presence of certain **lexico-syntactic patterns** can indicate a particular semantic relationship between two nouns.
Ex: *such as*

$NP\{,NP\}*\{,\}$ (and|or) other NP_H	...temples, treasuries, and other important civic buildings.
NP_H such as $\{NP,\}$* (or|and) NP	red algae such as Gelidium
such NP_H as $\{NP,\}$* (or|and) NP	works by such authors as Herrick, Goldsmith, and Shakespeare
NP_H $\{,\}$ including $\{NP,\}$* (or|and) NP	All common-law countries, including Canada and England
NP_H $\{,\}$ especially $\{NP,\}$* (or|and) NP	...most European countries, especially France, England, and Spain

Figure 20.14 Hand-built lexico-syntactic patterns for finding hypernyms (Hearst, 1992, 1998)

The coverage of such pattern-based method is limited by the number and accuracy of the available patterns. Unfortunately, once the obvious examples have been found, the process of creating patterns by hand becomes difficult and time consuming. As a solution, we have **bootrsapping** methods, common for info extraction.

Apart from bootsrapping we can also use **large lexical resources like WordNet** as a source training information, in which each WordNet hypernym/ hyponym pair tells us something about what kind of words are in this relations; and then train a classifier to help find new words that exhibit this relation.

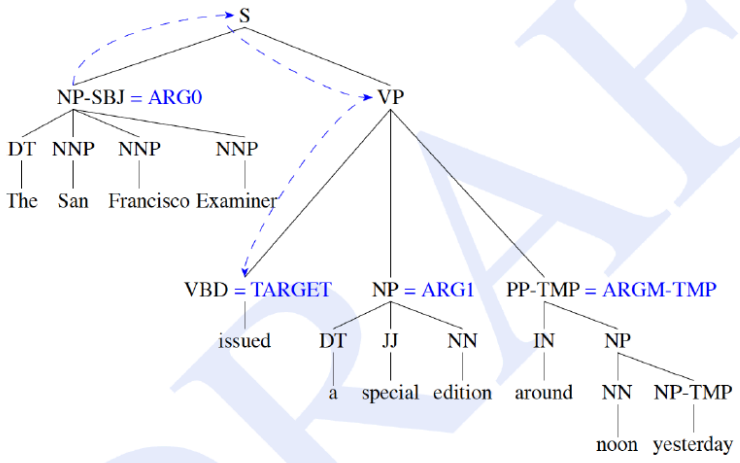
20.9 Semantic Role Labeling

Semantic role labeling is the task of **automatically finding the semantic roles for each predicate** in a sentence. More specifically, that means determining **which constituents in a sentence are semantic arguments for given predicate**, and then determining the appropriate role for each of those arguments.

[You] can't [blame] [the program] [for being unable to identify a processor]
COGNIZER TARGET EVALUEE REASON

[The San Francisco Examiner] issued [a special edition] [around noon yesterday]
ARG0 TARGET ARG1 ARGM-TMP

It starts with parsing. Then, the parsing result is traversed to find all predicate-being words. For each of these predicates, the tree is again traversed to determine which role, if any, each constituent in the parse plays with respect to the predicate. The algorithm makes this judgement by first characterizing the constituents as a set of features with respect to the predicate. A classifier trained on an appropriate training set is then passed this feature set and makes the appropriate assignment.



- **ADV** adverbial, default tag
- **BNE** beneficiary
- **CND** condition
- **DIR** direction
- **DGR** degree
- **EXT** extent
- **FRQ** frequency
- **LOC** locative
- **MNR** manner
- **PRP** purpose or reason
- **TMP** temporal
- **TPC** topic
- **CRD** coordinated arguments
- **PRD** predicate
- **PSR** possessor
- **PSE** possesses

20.10 Unsupervised Sense Disambiguation

It is expensive and difficult to build large corpora in which each word is labeled for its word sense. For this reason, unsupervised approaches to sense disambiguation are an exciting and important research area.

- Automatic creation of sense inventory based on training set
- Combines clustering and distance metric between vectors
- No label for sense
- Evaluation more difficult

Context vector:

$$\vec{w} = (f_1, f_2, f_3, \cdots, f_{1000})$$

01. Use context vector in training corpus in three steps

1. For each token w_i of word w in a corpus, compute a context vector $\vec{c}$.
2. Use a **clustering algorithm** to **cluster** these word token context vectors $\vec{c}$ into a predefined number of groups or clusters. Each cluster defines a sense of w.
3. Compute the **vector centroid** of each cluster. Each vector centroid $\vec{s}_j$ is a **sense vector** representing that sense of w.

02. Disambiguate a particular token t of w in three steps

1. Compute a context vector $\vec{c}$ for t as discussed above.
2. Retrieve all sense vectors s_j for w.
3. Assign t to the sense represented by the sense vector s_j that is closest to t.

WSD. A machine L task. We want to have a classifier.

What we want to define is what kind of info we provide to our classifier. That is the difficulty of WSD. Trying to decide what is a good information to give to the classifier.

For each word that you want to decide what is the meaning you need to design a new classifier. You get a really long vector with all the features.

There are a lot of design choices.

Naive bayes

(We have seen the before) This is a technique that we used before.

We want to find out the best sense of a word in the context. (we know what the context is, because it is represented with the long vector) and we want to find the sense that maximizes the probability.

Interested in the sense that maximizes the probability = how lonely it is that given this sense we find this context and multiplied by ..

When we put the PI is because: it is very unlikely to find the whole context (the vector with the 2 words before and after and their PoS tags) in a new text. Therefore, we assume that each word in the context is independent. Then we are not looking for the words in the exact same position repenting them in a new text but the words independently. We assume that the words do not have relationship between each other, which is not true but it works in the practice.

APPLICATIONS

Chapter 25. Machine Translation

1/10/18

During the second world war. Translate Russian messages.

MT approaches:

- Direct, transfer, interlingua
- SMT

Evaluation

Use of computers to automatically translate form one language to another. Translation requires a **deep understanding of the source** language and a **poetic and creative command of the target** language.

Fully automatic, high-quality translation (FAHQT) is far too hard to automate completely.
But we are good for tasks that a **rough translation** is ok, with **human post-editor**, or **limited small sublanguages domains** (ex. Weather).

It is hard because the **translation divergences** → differences between languages. An understanding of what cause them will help us in building models that overcome the differences.

Typology or systematic: - can be modeled in a general way
Languages have different structures
Universals describe regularities

- **Morphological**
  - Number of morphemes per word (ex: *in*, *come*, *-ing* → *incoming - 3 morphemes*)
    - **Isolating** languages→ 1 m/w - *Vietnamese*
    - **Polysynthetic** languages → A lot of m/w - *Eskimo*
  - Degree in which morphemes are segmentable
    - **Agglutinative** languages → clear boundaries - *Turkish*
    - **Fusion** languages → not clear boundaries - *Russian*
- **Syntactically**
  - **SVO** (subject - verb - object) - German, French, English
  - **SOV** - Hindi, Japanese
  - **VSO** - Irish, Arabic

- **Motion indicator**
  - **Verb-Framed**: mark the direction of the motion on the verb - *Spanish acercarse, alcanzar, entrar, salir*
  - **Satellite-framed**: mark the direction of the motion on the satellite - *English crawl out, float off, jump down, run after*

- **Referential density** (omit pronouns)
  - **Hot* languages**: make everything explicit - *English*
    [The boss] came upon a book. [He] showed it to a wondering decoder.
  - **Pro-drop** or **cold* languages** : can omit pronouns - *Spanish*
    [El jefe] dio con un libro. ø Mostró a un descifrador ambulante.

- **Adjective-Verb** vs **Verb-Adjective**

  English: Green witch

  Spanish: Bruja Verde

  French: Maison bleue

- **Other trivial aspects**

  DD/MM/YY *British English*

  MM/DD/YY *American English*

  YYMMDD *Japanese*

Lexical divergence or idiosyncratic: - must be dealt with one by one

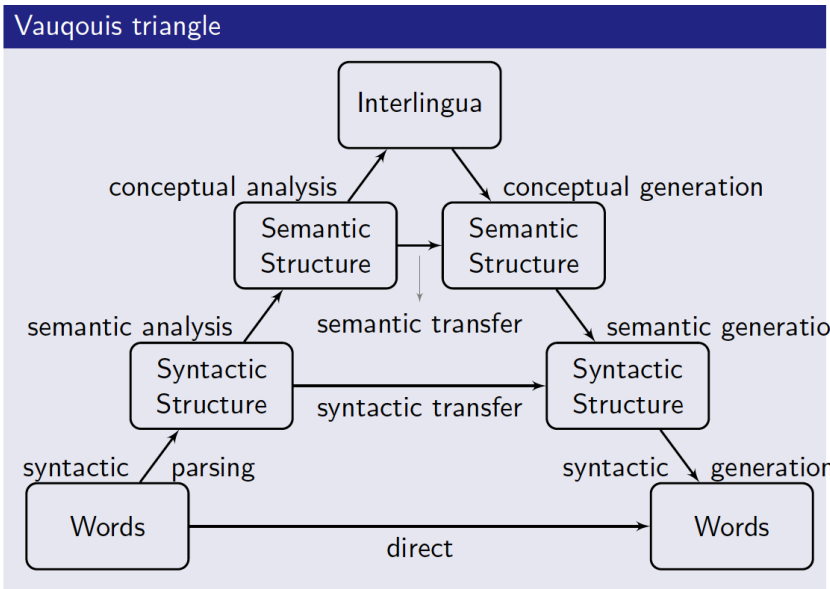
Meaning of words in two languages often only partially overlaps.
Translation requires solving the same problems as WSD.

English: I know he just bought a book // **French**: Je sais qu'il vient d'acheter un livre

English: I know John // **French**: Je connais Jean

In translation, we can think of disambiguation as a kind of **specification**: we have to make a vague word more specific in the target language.

25.2 Classical MT and the Vauquois Triangle



(more depth of analysis – less transfer knowedge)

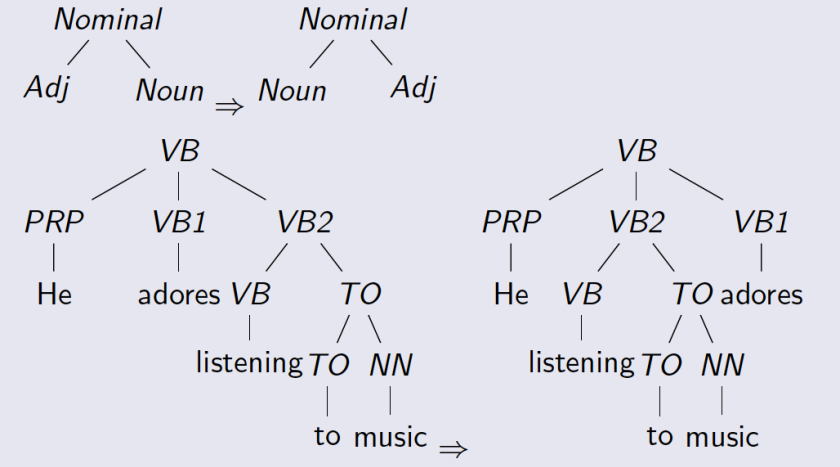
- **Direct Translation:** word by word. Downside: we need a large bilingual dictionary. Incremental transformation from source to target sentence.
  - Word by word
  - Based on a large bilingual dictionary
  - Simple reordering rules can be applied
  - Important: it has no parsing components nor any knowledge about phrasing or grammatical structure in the source or target language
  - Too focused on individual words

  Source text → Morph. Analysis → Lexical transfer → Local reordering → Morph. Generation → Target text

  Transfer approaches: 1rst parse source text. 2nd apply rules to transform the source-language parse into a target-language parse. 3rd generate the target language sentence from the parse tree. **Manipulate syntactic representations**

  Applies Contrastive Knowledge: knowledge about the differences between the two languages. It requires transfer rules for each pair of languages. An example of contrastive knowledge rules are:

  Example 1: From English to Spanish→ Consider reordering **adj** and **noun**
  Example 2: From English to Japanese → Consider the **SVO** and **SOV**



In practice, we need messy rules that combine rich lexical knowledge of both languages with syntactic and semantic features. We combine the direct and the transfer approaches, to combine **rich bilingual dictionaries** but also using **taggers and parsers.**

Transfer approaches are not optimal for the many-to-many translations as they require transfer rules for each pair of languages.

- **Interlingua approaches:** 1rst analyze the in source-language text into some abstract meaning representation (interlingua). 2nd generate into the target-language from this interlingua representation. **Extract the meaning** of the input and expressing it in the target language. – **create semantic analysis.**

  Deep semantic analysis but **no contrastive knowledge** needed. The amount of knowledge needed would be proportional to the number of languages the system handles, rather to the square of the number.

  Good for the many to many transitions. Just built the generation part for all the languages you want to translate into. It is really efficient.

  They tried a lot of things to do interlingua:
  - FOL? but that is not enough
  - Semantic representation?
  - What properties should the interlingua have?

  Summing up, the task would be like translating to interlingua and then translate from interlingua.
  It is **really difficult to find a good interlingua**. That is why they kind of gave up.

25.3 Statistical MT (Data based approach)

Rosetta stone: no dictionary, no grammar, no meaning, no context, just pairs of text.

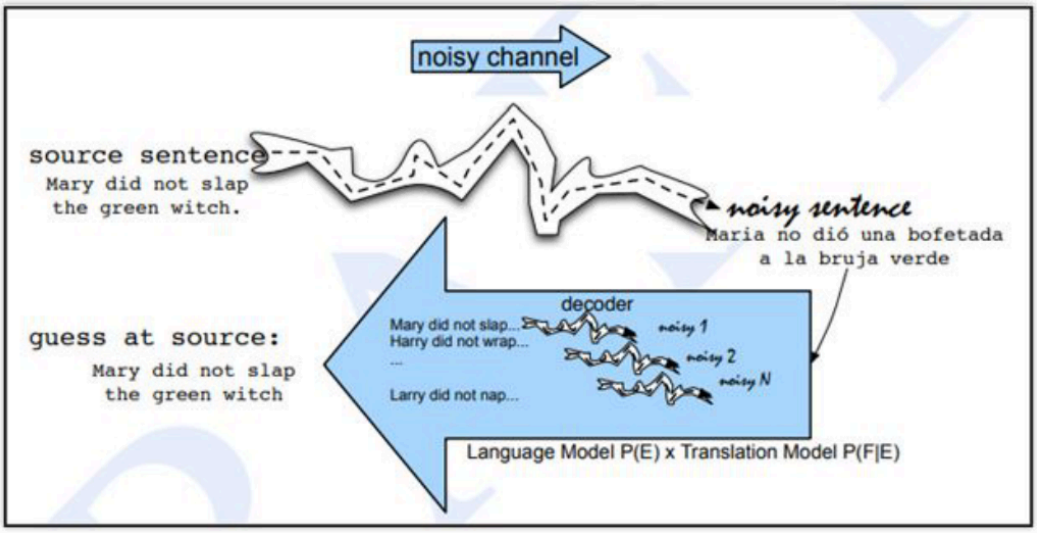
There is always a trade-off between **faithfulness** & **fluency**. Ex:

- "The Lord is for me like someone who looks after animals with cotton-like hair"
- "The Lord will watch over me"

Statistical MT maximizes over faithfulness & fluency.

best-translation T = $argmax_T$ *faithfulness(T,S) fluency(T)*

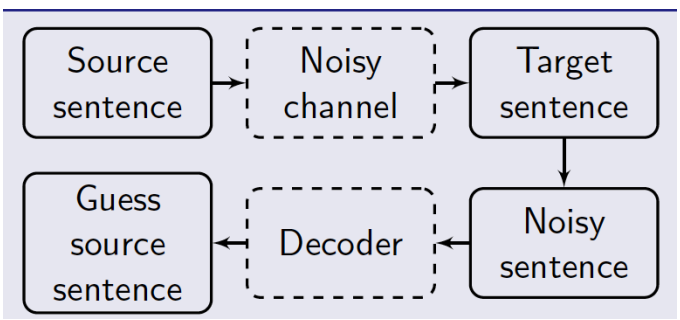
Bayesian Noisy Channel:



(we have seen it in the chapter 5 – PoS Tagging)

Noisy Chanel (phone) someone is speaking English on one end of the phone, on the other end, you hear French. You are trying to understand what happened in this channel to figure out was the original text. Complicated. Therefore, we reverse what is the target and what is the source.

Machine Translation is decoding. Decode using statistics.



P(E|F), rewrite in Bayes rule:

$$\hat{E} = \text{argmax}_E P(E|F)$$
$$= \text{argmax}_E P(F|E)P(E) / P(F)$$
$$= \text{argmax}_E P(F|E)P(E)$$

The noisy channel requires three components:

1. **Language model** to compute P(E) --- monolingual
    a. Given a string e, give P(e)
    b. Good string → high P(e)
    c. Random string → low P(e)

2. **Translation model** to compute P (F | E)
    a. Given pair of strings [f, e], give P (f | e)
    b. If [f, e] look like translation → high P (f | e)
    c. If [f, e] do not look like translation → low P (f | e)

3. **Decoder**, which is given F and produces the most probable E
    a. Given LM, TM, sentence f find e:
    b. argmax_e P (e) * P (f | e)

Statistical MT are based on **N-grams** language models.

25.4 Translation model - P (f | e)

P (E|F) we want to know the English sentence that maximizes the probability here.
Given the French sentence, what is the probability of the English. → by checking all the English sentences that can occur. There are a lot of English sentences. So, what we need to find is words or phrases that can translate into English.

Decoding (same as lecture 4 POS) - try out all the different combinations to find the best translations.

Finding the alignments: try all the possible alignments and see which ones appear more often.