Introduction to Computational Linguistics

Wiebke Petersen

Heinrich-Heine-Universität Düsseldorf
Institute of Language and Information
Computational Linguistics
www.phil-fak.uni-duesseldorf.de/~petersen/

NLL Riga, 28th November - 1st December 2008

Part I

Introduction

Outline

Common names

- Computational Linguistics (CL)
- Natural Language Processing (NLP)
- Language Engineering
- Human Language Technology (HLT)

computational linguistics (broad sense): interdisciplinary research field (between linguistics and computer science) which develops concrete algorithms for natural language processing (machine translation, machine speech recognition ...)

computational linguistics (narrow sense): discipline in modern linguistics which develops, implements and investigates computational models of human language.

Theoretical CL (Uszkoreit: What is CL?)

- Theoretical CL takes up issues in theoretical linguistics and cognitive science.
- It deals with formal theories about the linguistic knowledge that a human needs for generating and understanding language
- Computational linguists develop formal models simulating aspects of the human language faculty and implement them as computer programmes.

Applied CL (Uszkoreit: What is CL?)

- Applied CL focusses on the practical outcome of modeling human language use. (other terms: HLT, NLP)
- The goal is to create software products that have some knowledge of human language.
- Such products are going to change our lives. They are urgently needed for improving human-machine interaction since the main obstacle in the interaction between human and computer is a communication problem, the use of human language can increase the acceptance of software and the productivity of its users.

advanced NLP applications

- dialogue systems / conversational agents
  - simplifies human-computer interaction
- machine translation
  - simplifies human-human interaction
- question answering
  - simplifies usage of the web

advanced NLP applications

- dialogue systems / conversational agents
  - simplifies human-computer interaction
- machine translation
  - simplifies human-human interaction
- question answering
  - simplifies usage of the web

simpler NLP applications

- spell checking
- grammar checking
- word count

machine translation

state of the art

http://translate.google.com/translate_t

source Computational linguistics is an interdisciplinary field dealing with the statistical and rule-based modeling of natural language from a computational perspective.

target Datorlingvistika ir starpdisciplinārā jomā nodarbojas ar statistikas un uz likumu balstītas modelēšanas dabas valodu no skaitlošanas viedokla.

machine translation

Lidziga sun you bring us days,
Wisdom verige long you provide.
Celdamas itself ever higher,
People put you in higher take off.

Latvia and the Latvian celebrity prettiness,
Arts and the Knowledge refuge there.
Unfamiliar to the oak trees indefinitely showing no
All as the eternal fire.

machine translation

Lidziga sun you bring us days,
Wisdom verige long you provide.
Celdamas itself ever higher,
People put you in higher take off.

Latvia and the Latvian celebrity prettiness,
Arts and the Knowledge refuge there.
Unfamiliar to the oak trees indefinitely showing no
All as the eternal fire.

Lidziga saulei Tu atnes mums dienu,
Gudribu verigiem gariem Tu sniedz.
Celdamas augstaku pati arvienu,
Tautai Tu augstaku pacelties liec.

Latvijas slava un Latvijas glitums,
Makslam un zinibam patverums tur.
Svess lai, ka ozoliem muzigiem, vitums
Visiem, kas muzigu uguni kur.

Anthem "Latvijas Universitatei"

Sometimes human “translations” go wrong too!

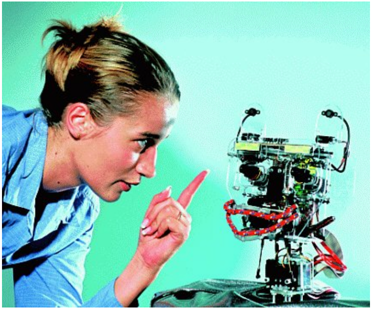
Welsh text reads: “I am not in the office at the moment. Send any work to be translated.”

question answering

possible questions

- What does "divergent" mean?
- What year was Abraham Lincoln born?
- How many states were in the United States that year?
- What do scientists think about the ethics of human cloning?
- What is the connection between CL and NLP?
- Who is the rector of the university of Riga?
- How far is Berlin from Riga?
- What kind of language is Latvian?

conversational agents



conversational agents

Interaction with HAL 9000 the computer in Stanley Kubrick's film "2001: A Space Odyssey":

Dave Bowman: Open the pod bay doors, HAL.

HAL: I'm sorry Dave, I'm afraid I can't do that.

required language knowledge

- speech recognition
- natural language understanding
- natural language generation
- speech synthesis

http://www-306.ibm.com/software/pervasive/tech/demos/tts.shtml

Knowledge needed to build HAL?

- Speech recognition and synthesis
  - Dictionaries (how words are pronounced)
  - Phonetics (how to recognize/produce each sound of English)
- Natural language understanding
  - Knowledge of the English words involved
    - What they mean
    - How they combine (what is a `pod bay door'?)
  - Knowledge of syntactic structure
    - I'm I do, Sorry that afraid Dave I'm can't

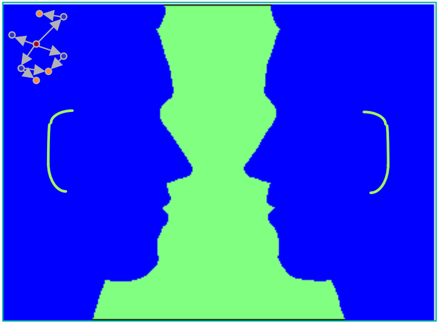
What’s needed?

- Dialog and pragmatic knowledge
  - “open the door” is a REQUEST (as opposed to a STATEMENT or information-question)
  - It is polite to respond, even if you’re planning to kill someone.
  - It is polite to pretend to want to be cooperative (I’m afraid, I can’t…)
  - What is `that’ in `I can’t do that’?
- Even a system to book airline flights needs much of this kind of knowledge

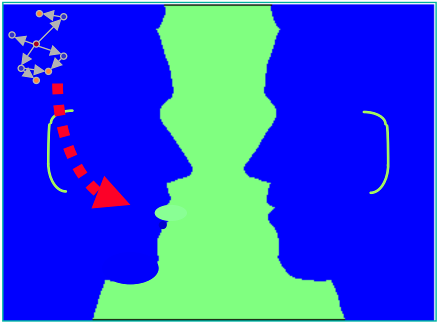
fascination language

- Language is an ability which is special to humans
- Humans are able to express and understand complex thoughts in seconds.
- Children are able to learn language within a few years.

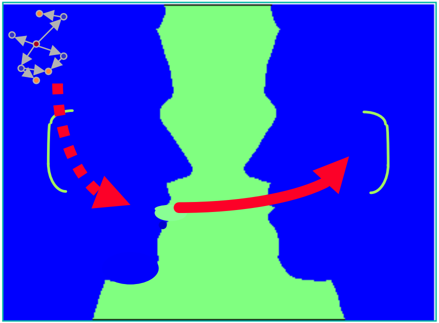
verbal communication



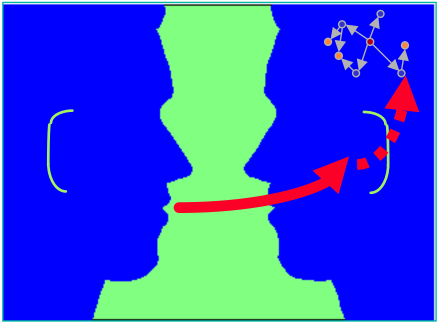
verbal communication



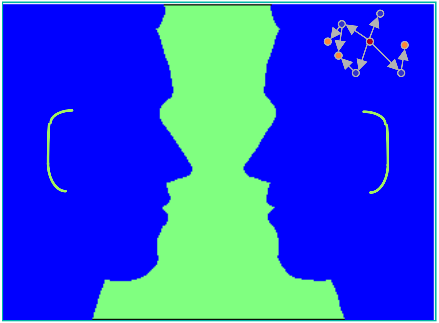
verbal communication



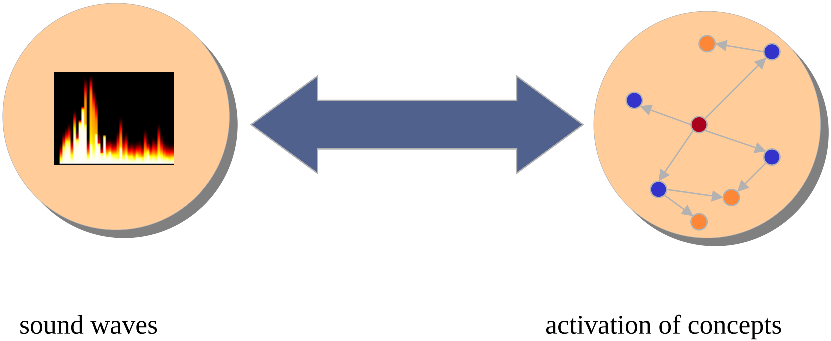
verbal communication



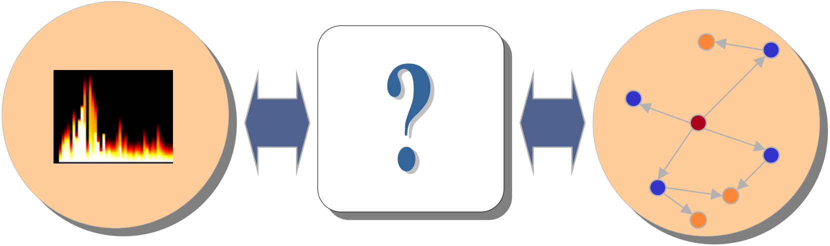
verbal communication



grammar



grammar



sound waves

grammar

activation of concepts

complexity of language

- Latvian, German, English, Chinese, ...

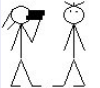
complexity of language

- Latvian, German, English, Chinese, . . .
- vague, ambiguous,

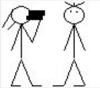
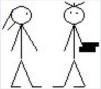
complexity of language

- Latvian, German, English, Chinese, . . .
- vague, ambiguous,
- ambiguities:
  - lexical ambiguities (call me tomorrow - the call of the beast)

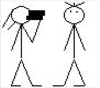
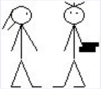
complexity of language

- Latvian, German, English, Chinese, . . .
- vague, ambiguous,
- ambiguities:
  - lexical ambiguities (call me tomorrow - the call of the beast)
  - structural ambiguities:
    - the woman sees the man with the binoculars 

complexity of language

- Latvian, German, English, Chinese, . . .
- vague, ambiguous,
- ambiguities:
  - lexical ambiguities (call me tomorrow - the call of the beast)
  - structural ambiguities:
    - the woman sees the man with the binoculars 
    - the woman sees the man with the binoculars 

complexity of language

- Latvian, German, English, Chinese, . . .
- vague, ambiguous,
- ambiguities:
  - lexical ambiguities (call me tomorrow - the call of the beast)
  - structural ambiguities:
    - the woman sees the man with the binoculars 
    - the woman sees the man with the binoculars 
- only experts: humans
- natural languages develop

Ambiguity

- Find at least 5 meanings of this sentence:
  - I made her duck

Ambiguity

- Find at least 5 meanings of this sentence:
  - I made her duck
- I cooked waterfowl for her benefit (to eat)
- I cooked waterfowl belonging to her
- I created the (plaster?) duck she owns
- I caused her to quickly lower her head or body
- I waved my magic wand and turned her into undifferentiated waterfowl
- At least one other meaning that's inappropriate for gentle company.

Ambiguity is Pervasive

- I caused her to quickly lower her head or body
  - **Lexical category**: “duck” can be a N or V
- I cooked waterfowl belonging to her.
  - **Lexical category:** “her” can be a possessive (“of her”) or dative (“for her”) pronoun
- I made the (plaster) duck statue she owns
  - **Lexical Semantics:** “make” can mean “create” or “cook”

Ambiguity is Pervasive

- **Grammar**: Make can be:
  - **Transitive: (verb has a noun direct object)**
    – I cooked [waterfowl belonging to her]
  - **Ditransitive: (verb has 2 noun objects)**
    – I made [her] (into) [undifferentiated waterfowl]
  - **Action-transitive (verb has a direct object and another verb)**
  - I caused [her] [to move her body]

Ambiguity is Pervasive

- **Phonetics!**
  - I mate or duck
  - I'm eight or duck
  - Eye maid; her duck
  - Aye mate, her duck
  - I maid her duck
  - I'm aid her duck
  - I mate her duck
  - I'm ate her duck
  - I'm ate or duck
  - I mate or duck

Exercise: Introduction

Exercise 1

- *Experiment on the following machine translators (e.g., Latvian – English, English – Latvian)*
  http://translate.google.com/translate_t
  http://babelfish.altavista.com/
  - *Try to identify problematic structures which result in faulty translations*
  - *Try to find reasons for the translation problems*
- *Experiment on the following question answering systems*
  http://www.ask.com/
  http://start.csail.mit.edu/
  - *Compare the systems*
  - *Which kind of question is answered adequately?*
  - *Which kind of question cannot be answered by the systems?*

Part II

Formal Languages (Introduction)

Outline

4 Preliminaries: sets

5 Alphabets and words

6 formal languages

sets

Georg Cantor (1845-1918)

By a set we mean any collection M into a whole of definite, distinct objects x (which are called the elements of M) of our perception or of our thought.
Two sets are equal iff they have precisely the same members.
The empty set $\emptyset$ is the set which has no elements.

notation

- $x \in M$: x is an element of set M.
- $M \subset N$: set M is a subset of set N, i.e., every element of set M is an element of set N.

notation

- $x \in M$: x is an element of set M.
- $M \subset N$: set M is a subset of set N, i.e., every element of set M is an element of set N.

set description

extensional set description $\{a_1, a_2, \ldots, a_n\}$ is the set which has the elements $a_1, a_2, \ldots, a_n$.
Example: $\{2, 3, 4, 5, 6, 7\}$

notation

- $x \in M$: x is an element of set M.
- $M \subset N$: set M is a subset of set N, i.e., every element of set M is an element of set N.

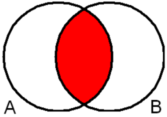
set description

extensional set description $\{a_1, a_2, \ldots, a_n\}$ is the set which has the elements $a_1, a_2, \ldots, a_n$.
Example: $\{2, 3, 4, 5, 6, 7\}$

intensional set description $\{x|A\}$ is the set consisting of all elements x which fulfill statement A.
Example: $\{x|x \in \mathbb{N} \text{ and } x < 8 \text{ and } 1 < x\}$

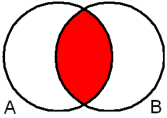
operations on sets

intersection: $A \cap B$

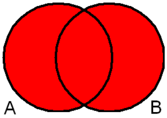


operations on sets

intersection: $A \cap B$

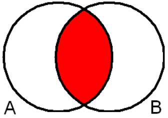


union: $A \cup B$

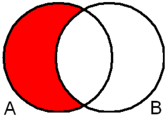


operations on sets

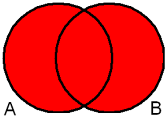
intersection: $A \cap B$



difference: $A \setminus B$

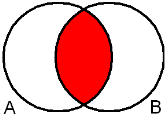


union: $A \cup B$

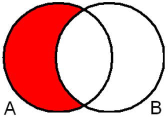


operations on sets

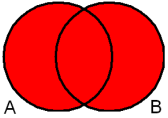
intersection: $A \cap B$



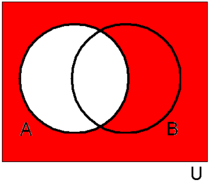
difference: $A \setminus B$



union: $A \cup B$



complement (in U): $C_U(A)$



Alphabets and words

Definition

- *alphabet* Σ*: nonempty, finite set of symbols*
- *word: a finite string* $x_1 \dots x_n$ *of symbols.*
- *length of a word* $|w|$*: number of symbols of a word* w *(example:* $|abbaca| = 6$*)*
- *empty word* ϵ*: the word of length* 0

Alphabets and words

Definition

- *alphabet* Σ: *nonempty, finite set of symbols*
- *word: a finite string* $x_1 \dots x_n$ *of symbols.*
- *length of a word* $|w|$: *number of symbols of a word* w *(example:* $|abbaca| = 6$*)*
- *empty word* ϵ: *the word of length* 0
- Σ^* *is the set of all words over* Σ
- Σ^+ *is the set of all nonempty words over* Σ *(*$\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$*)*

Exercise: alphabets and words

Exercise 2

Let $\Sigma = \{a, b, c\}$*:*

- *Write down a word of length* 4*.*
- *Which of the following expressions is a word and of what length is it:*
  'aa', 'caab', 'da'
- *What is the difference between* Σ^* *and* Σ^+*?*
- *How many elements do* Σ^* *and* Σ^+ *have?*

Operations on words: Concatenation

Definition

The concatenation of two words $w = a_1 a_2 \ldots a_n$ *and* $v = b_1 b_2 \ldots b_m$ *with* $n, m \geq 0$ *is*

$$w \circ v = a_1 \ldots a_n b_1 \ldots b_m$$

Sometimes we write uv *instead of* $u \circ v$.

Operations on words: Concatenation

Definition

The concatenation of two words $w = a_1 a_2 \dots a_n$ and $v = b_1 b_2 \dots b_m$ with $n, m \geq 0$ is

$$w \circ v = a_1 \dots a_n b_1 \dots b_m$$

Sometimes we write uv instead of $u \circ v$.

$w \circ \epsilon = \epsilon \circ w = w$ neutral element

$u \circ (v \circ w) = (u \circ v) \circ w$ associativity

Operations on words: exponents and reversals

Exponents

- w^n: w concatenated n-times with itself.
- $w^0 = \epsilon$: w concatenated '0-times' with itself.

Operations on words: exponents and reversals

Exponents

- w^n: w concatenated n-times with itself.
- $w^0 = \epsilon$: w concatenated '0-times' with itself.

Reversals

- The reversal of a word w is denoted w^R (example: $(abcd)^R = dcba$.
- A word w with $w = w^R$ is called a palindrome.

(madam, mum, otto, anna,. . .)

Exercise: Operations on words

Exercise 3

If $w = aabc$ and $v = bcc$ are words, evaluate:

- $w \circ v$
- $((w^R \circ v)^R)^2$
- $w \circ (v^R \circ w^3)^0$

Formal language

Definition

A formal language L is a set of words over an alphabet Σ.

Formal language

Definition

A formal language L is a set of words over an alphabet Σ.

Examples:

- language L_{pal} of the palindromes in English
  $L_{pal} = \{\text{mum, madam, } \dots\}$

Formal language

Definition

A *formal language* *L is a set of words over an alphabet Σ.*

Examples:

- language L_{pal} of the palindromes in English
  $L_{pal} = \{\text{mum}, \text{madam}, \dots\}$
- language L_{Mors} of the letters of the latin alphabet encoded in the Morse code: $L_{Mors} = \{\cdot -, - \cdot \cdot\cdot, \dots, - - \cdot\cdot\}$

Formal language

Definition

A *formal language* *L is a set of words over an alphabet Σ.*

Examples:

- language L_{pal} of the palindromes in English
  $L_{pal} = \{\text{mum}, \text{madam}, \dots\}$
- language L_{Mors} of the letters of the latin alphabet encoded in the Morse code: $L_{Mors} = \{\cdot-, -\cdot\cdot\cdot, \dots, --\cdot\cdot\}$
- the empty set

Formal language

Definition

A formal language L is a set of words over an alphabet Σ.

Examples:

- language L_{pal} of the palindromes in English
  $L_{pal} = \{\text{mum, madam, } \dots\}$
- language L_{Mors} of the letters of the latin alphabet encoded in the Morse code: $L_{Mors} = \{\cdot -, - \cdot \cdot\cdot, \dots, - - \cdot\cdot\}$
- the empty set
- the set of words of length 13 over the alphabet $\{a, b, c\}$

Formal language

Definition

A *formal language* *L is a set of words over an alphabet Σ.*

Examples:

- language L_{pal} of the palindromes in English
  $L_{pal} = \{\text{mum, madam, } \dots\}$
- language L_{Mors} of the letters of the latin alphabet encoded in the Morse code: $L_{Mors} = \{\cdot -, - \cdot \cdot\cdot, \dots, - - \cdot\cdot\}$
- the empty set
- the set of words of length 13 over the alphabet $\{a, b, c\}$
- English?

Describing formal languages by enumerating all words

- Peter says that Mary has fallen off the tree.
- Oskar says that Peter says that Mary has fallen off the tree.
- Lisa says that Oskar says that Peter says that Mary has fallen off the tree.
- ...

Describing formal languages by enumerating all words

- Peter says that Mary has fallen off the tree.
- Oskar says that Peter says that Mary has fallen off the tree.
- Lisa says that Oskar says that Peter says that Mary has fallen off the tree.
- ...

The set of strings of a natural language is infinite.

The enumeration does not gather generalizations.

Describing formal languages by grammars

Grammar

- A formal grammar is a generating device which can generate (and analyze) strings/words.
- Grammars are finite rule systems.
- The set of all strings generated by a grammar is the formal language generated by the grammar.

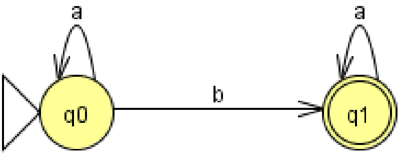
$$\begin{array}{rclrclrcl} S & \rightarrow & NP\ VP & VP & \rightarrow & V & NP & \rightarrow & D\ N \\ D & \rightarrow & \text{the} & N & \rightarrow & \text{cat} & V & \rightarrow & \text{sleeps} \end{array}$$

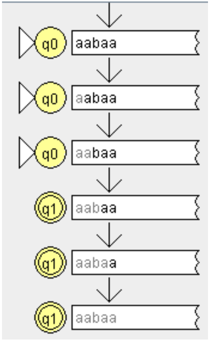
Generates: the cat sleeps

Describing formal languages by automata

Automaton

- An automaton is a recognizing device which accepts strings/words.
- The set of all strings accepted by an automaton is the formal language accepted by the automaton.





Language concatenation

Definition

- *The* *concatenation* *of K and L is the formal language:*

$$K \circ L := \{v \circ w \in \Sigma^* | v \in K, w \in L\}$$

Language concatenation

Definition

- *The concatenation of K and L is the formal language:*

$$K \circ L := \{v \circ w \in \Sigma^* | v \in K, w \in L\}$$

- $L^n = \underbrace{L \circ L \circ L \ldots \circ L}_{n\text{-times}}$
- $L^* := \bigcup_{n \geq 0} L^n$. *Note:* $\epsilon \in L^*$ *for any language* L.

Language concatenation

Example 1

$K = \{abb, a\}$ and $L = \{bbb, ab\}$

- $K \circ L =$

Language concatenation

Example 1

$K = \{abb, a\}$ and $L = \{bbb, ab\}$

- $K \circ L = \{abbbbb, abbab, abbb, aab\}$ and
  $L \circ K =$

Language concatenation

Example 1

$K = \{abb, a\}$ and $L = \{bbb, ab\}$

- $K \circ L = \{abbbbb, abbab, abbb, aab\}$ and
  $L \circ K = \{bbbabb, bbba, ababb, aba\}$
- $K \circ \emptyset =$

Language concatenation

Example 1

$K = \{abb, a\}$ and $L = \{bbb, ab\}$

- $K \circ L = \{abbbbb, abbab, abbb, aab\}$ and
  $L \circ K = \{bbbabb, bbba, ababb, aba\}$
- $K \circ \emptyset = \emptyset$
- $K \circ \{\epsilon\} =$

Language concatenation

Example 1

$K = \{abb, a\}$ and $L = \{bbb, ab\}$

- $K \circ L = \{abbbbb, abbab, abbb, aab\}$ and
  $L \circ K = \{bbbabb, bbba, ababb, aba\}$
- $K \circ \emptyset = \emptyset$
- $K \circ \{\epsilon\} = K$
- $K^2 =$

Language concatenation

Example 1

$K = \{abb, a\}$ and $L = \{bbb, ab\}$

- $K \circ L = \{abbbbb, abbab, abbb, aab\}$ and
  $L \circ K = \{bbbabb, bbba, ababb, aba\}$
- $K \circ \emptyset = \emptyset$
- $K \circ \{\epsilon\} = K$
- $K^2 = \{abbabb, abba, aabb, aa\}$

Exercise: formal languages

Exercise 4

If $K = \{aa, aaaa, ab\}$ *and* $L = \{bb, aa\}$ *are languages, evaluate*

1. $K \circ L$
2. $L \circ K$
3. $\{\epsilon\} \circ L$
4. $\{\epsilon\} \circ \emptyset$
5. $K \circ \emptyset$
6. K^3
7. $K \setminus L$

Part III

Finite State Automatons and Regular Languages

Outline

7. regular expressions

8. finite state automatons

Regular expressions

RE: syntax

The set of regular expressions RE_Σ over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ is defined by:

- $\underline{\emptyset}$ is a regular expression.
- ϵ is a regular expression.
- $a_1, \ldots, a_n$ are regular expressions
- If a and b are regular expressions over Σ then
  - $(a + b)$
  - $(a \bullet b)$
  - $(a^\star)$

  are regular expressions too.

(The brackets are frequently omitted w.r.t. the following dominance scheme: $\star$ dominates $\bullet$ dominates $+$)

Regular expressions

RE: semantics

Each regular expression r over an alphabet Σ describes a formal language $L(r) \subseteq \Sigma^*$.
Regular languages are those formal languages which can be described by a regular expression.
The function L is defined inductively:

- $L(\underline{\emptyset}) = \emptyset$, $L(\epsilon) = \{\epsilon\}$, $L(a_i) = \{a_i\}$
- $L(a + b) = L(a) \cup L(b)$
- $L(a \bullet b) = L(a) \circ L(b)$
- $L(a^\star) = L(a)^*$

Exercise: regular expressions

Exercise 5

Find a regular expression which describes the regular language L (be careful: at least one language is not regular!)

- *L is the language over the alphabet* $\{a, b\}$ *with* $L = \{aa, \epsilon, ab, bb\}$.
- *L is the language over the alphabet* $\{a, b\}$ *which consists of all words which start with a nonempty string of a's followed by any number of b's*
- *L is the language over the alphabet* $\{a, b\}$ *such that every a has a b immediately to the right.*
- *L is the language over the alphabet* $\{a, b\}$ *which consists of all words which contain an even number of a's.*
- *L is the language of all palindromes over the alphabet* $\{a, b\}$.

What we know so far about formal languages

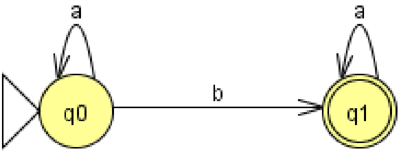
- Formal languages are sets of words (NL: sets of sentences) which are strings of symbols (NL: words).
- Everything in the set is a "grammatical word", everything else isn't.
- Some formal languages, namely the regular ones, can be described by regular expressions
  Example: $(a^\star \bullet b \bullet a^\star \bullet b \bullet a^\star)^\star$ is the regular language consisting of all words over the alphabet $\{a, b\}$ which contain an even number of b's.
- Not all formal languages are regular (We have not proven this yet!).
  Example: The formal language of all palindromes over the alphabet $\{a, b\}$ is not regular.

Deterministic finite-state automaton (DFSA)

Definition

A deterministic finite-state automaton is a tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ *with:*

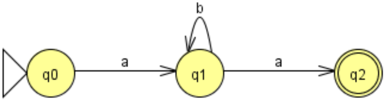
1. *a finite, non-empty set of states* Q
2. *an alphabet* Σ *with* $Q \cap \Sigma = \emptyset$
3. *a partial transition function* $\delta : Q \times \Sigma \to Q$
4. *an initial state* $q_0 \in Q$ *and*
5. *a set of final/accept states* $F \subseteq Q$.



accepts: $L(a^\star ba^\star)$

partial/total transition function

FSA with partial transition function



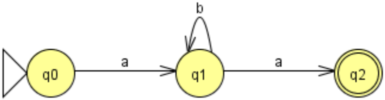
accepts $ab^{\star}a$

	a	b
q0	q1	
q1	q2	q1
q2		

transition table

partial/total transition function

FSA with partial transition function

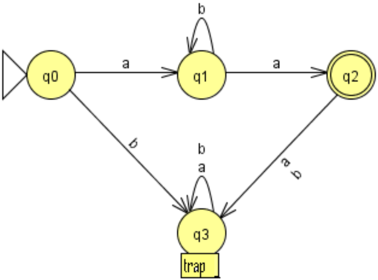


accepts $ab^{\star}a$

	a	b
q0	q1	
q1	q2	q1
q2		

transition table

FSA with complete transition function



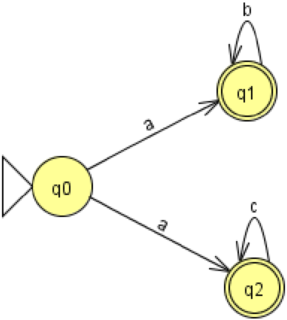
accepts $ab^{\star}a$

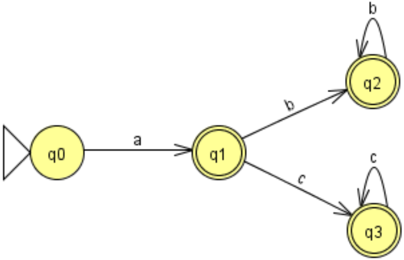
	a	b
q0	q1	q3
q1	q2	q1
q2	q3	q3
q3	q3	q3

transition table

Example DfSA / NDFSA

The language $L(ab^\star + ac^\star)$ is accepted by





Nondeterministic finite-state automaton NDFSA

Definition

A nondeterministic finite-state automaton is a tuple $\langle Q, \Sigma, \Delta, q_0, F\rangle$ *with:*

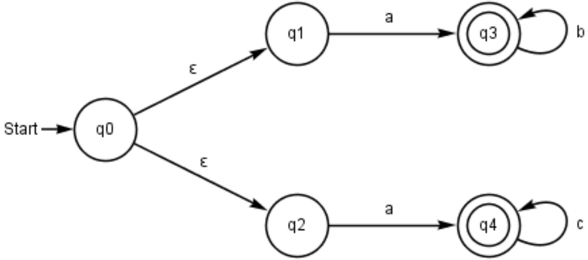
1. *a finite non-empty set of states* Q
2. *an alphabet* Σ *with* $Q \cap \Sigma = \emptyset$
3. **a transition relation** $\Delta \subseteq Q \times \Sigma \times Q$
4. *an initial state* $q_0 \in Q$ *and*
5. *a set of final states* $F \subseteq Q$.

Theorem

A language L can be accepted by a DFSA iff L can be accepted by a NFSA.

Note: Even automatons with ϵ-transitions accept the same languages like NDFSA's.

Automaton with ϵ-transition



Exercise 6

Give an FSA for each of the following languages over the alphabet $\{a, b\}$ *(and try to make it deterministic):*

- $L = \{w|$ *between each two 'b's in w there are at least two 'a's*$\}$
- $L = \{w|w$ *is any word except "ab"*$\}$
- $L = \{w|w$ *does not contain the infix "ba"*$\}$
- $L = \{w|w$ *contains at most three 'b's*$\}$
- $L = \{w|w$ *contains an even number of 'a's*$\}$
- $L((a^\star b)^\star ab^\star)$
- $L(a^\star(bb)^\star)$
- $L(ab^\star b)$.
- $L((ab^\star + ba^\star a))$

Finite-state automatons accept regular languages

Theorem (Kleene)

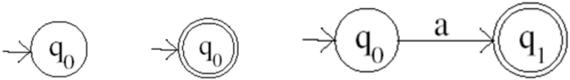
Every language accepted by a DFSA is regular and every regular language is accepted by some DFSA.

Finite-state automatons accept regular languages

Theorem (Kleene)

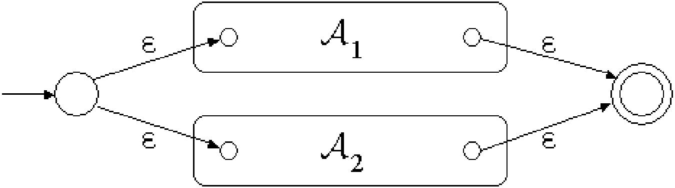
Every language accepted by a DFSA is regular and every regular language is accepted by some DFSA.

proof idea (one direction): Each regular language is accepted by a

NDFSA: 

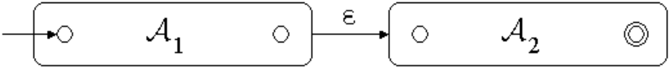
Proof of Kleene's theorem (cont.)

If R_1 and R_2 are two regular expressions such that the languages $L(R_1)$ and $L(R_2)$ are accepted by the automatons $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, then $L(R_1 + R_2)$ is accepted by:



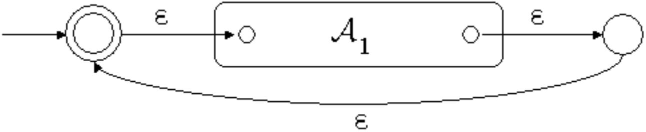
Proof of Kleene's theorem (cont.)

$L(R_1 \bullet R_2)$ is accepted by:



Proof of Kleene's theorem (cont.)

$L(R_1^*)$ is accepted by:



Closure properties of regular languages

Theorem

1. *If L_1 and L_2 are two regular languages, then*
   - *the union of L_1 and L_2 ($L_1 \cup L_2$) is a regular language too.*
   - *the intersection of L_1 and L_2 ($L_1 \cap L_2$) is a regular language too.*
   - *the concatenation of L_1 and L_2 ($L_1 \circ L_2$) is a regular language too.*
2. *The complement of every regular language is a regular language too.*
3. *If L is a regular language, then L^* is a regular language too.*

Exercise 7

Prove the theorem.

Pumping lemma for regular languages

Lemma (Pumping-Lemma)

If L is an infinite regular language over Σ, then there exists words $u, v, w \in \Sigma^*$ such that $v \neq \epsilon$ and $uv^iw \in L$ for any $i \geq 0$.

proof sketch:

Pumping lemma for regular languages

Lemma (Pumping-Lemma)

If L is an infinite regular language over Σ, then there exists words $u, v, w \in \Sigma^*$ such that $v \neq \epsilon$ and $uv^iw \in L$ for any $i \geq 0$.

proof sketch:

- Any regular language is accepted by a DFSA with a finite number n of states.

Pumping lemma for regular languages

Lemma (Pumping-Lemma)

If L is an infinite regular language over Σ, then there exists words $u, v, w \in \Sigma^*$ such that $v \neq \epsilon$ and $uv^iw \in L$ for any $i \geq 0$.

proof sketch:

- Any regular language is accepted by a DFSA with a finite number n of states.
- Any infinite language contains a word z which is longer than n ($|z| \geq n$).

Pumping lemma for regular languages

Lemma (Pumping-Lemma)

If L is an infinite regular language over Σ, then there exists words $u, v, w \in \Sigma^*$ such that $v \neq \epsilon$ and $uv^iw \in L$ for any $i \geq 0$.

proof sketch:

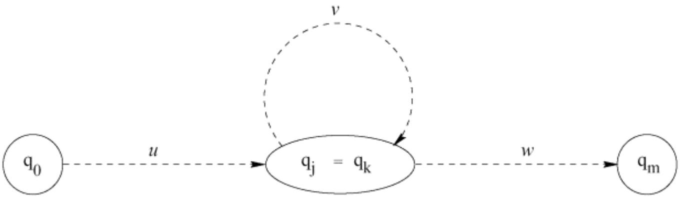
- Any regular language is accepted by a DFSA with a finite number n of states.
- Any infinite language contains a word z which is longer than n ($|z| \geq n$).
- While reading in z, the DFSA passes at least one state q_j twice.

Pumping lemma for regular languages (cont.)

Lemma (Pumping-Lemma)

If L is an infinite regular language over Σ, then there exists words $u, v, w \in \Sigma^*$ such that $v \neq \epsilon$ and $uv^iw \in L$ for any $i \geq 0$.

proof sketch:



$L = \{a^n b^n \ : \ n \geq 0\}$ is not regular

- $L = \{a^n b^n \ : \ n \geq 0\}$ is infinite.
- Suppose L is regular. Then there exists $u, v, w \in \{a, b\}^*$, $v \neq \epsilon$ with $uv^n w \in L$ for any $n \geq 0$.
- We have to consider 3 cases for v.

$L = \{a^n b^n \ : \ n \geq 0\}$ is not regular

- $L = \{a^n b^n \ : \ n \geq 0\}$ is infinite.
- Suppose L is regular. Then there exists $u, v, w \in \{a, b\}^*$, $v \neq \epsilon$ with $uv^n w \in L$ for any $n \geq 0$.
- We have to consider 3 cases for v.
  1. v consists of a's and b's.

$L = \{a^n b^n \ : \ n \geq 0\}$ is not regular

- $L = \{a^n b^n \ : \ n \geq 0\}$ is infinite.
- Suppose L is regular. Then there exists $u, v, w \in \{a, b\}^*$, $v \neq \epsilon$ with $uv^n w \in L$ for any $n \geq 0$.
- We have to consider 3 cases for v.
  1. v consists of a's and b's.
  2. v consists only of a's.

$L = \{a^n b^n \ : \ n \geq 0\}$ is not regular

- $L = \{a^n b^n \ : \ n \geq 0\}$ is infinite.
- Suppose L is regular. Then there exists $u, v, w \in \{a, b\}^*$, $v \neq \epsilon$ with $uv^n w \in L$ for any $n \geq 0$.
- We have to consider 3 cases for v.
    1. v consists of a's and b's.
    2. v consists only of a's.
    3. v consists only of b's.

Exercise: pumping lemma

Exercise 8

Are the following languages regular?

1. $L_1 = \{w \in \{a,b\}^* : w \text{ contains an even number of } b's\}$.
2. $L_2 = \{w \in \{a,b\}^* : w \text{ contains as many } b's \text{ as } a's\}$.
3. $L_3 = \{ww^R \in \{a,b\}^* : ww^R \text{ is a palindrome over } \{a,b\}^*\}$.

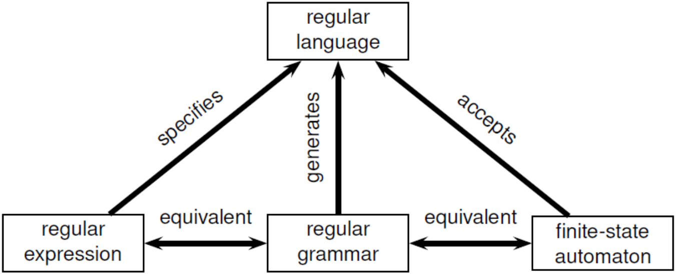
Intuitive rules for regular languages

- L is regular if it is possible to check the membership of a word simply by reading it symbol for symbol while using only a finite stack.

Intuitive rules for regular languages

- L is regular if it is possible to check the membership of a word simply by reading it symbol for symbol while using only a finite stack.
- Finite-state automatons are too weak for:
    - counting in $\mathbb{N}$ (“same number as”);
    - recognizing a pattern of arbitrary length (“palindrome”);
    - expressions with brackets of arbitrary depth.

Summary: regular languages



Prolog: the basics

- facts: state things that are unconditionally true of the domain of interest.
  `human(sokrates).`
- rules: relate facts by logical implications.
  `mortal(X) :- human(X).`
  - head: left hand side of a rule
  - body: right hand side of a rule
  - clause: rule or fact.
  - predicate: collection of clauses with identical heads.
- knowledge base: set of facts and rules
- queries: make the Prolog inference engine try to deduce a positive answer from the information contained in the knowledge base.
  `?- mortal(sokrates).`

Prolog: some syntax

- facts: `fact.`
- rules: `head :- body.`
- conjunction: `head :- info1 , info2.`
- atoms start with small letters
- variables start with capital letters

Exercise: `father(X,Y) :- parent(X,Y), male(X).`

lists in Prolog

- Lists are recursive data structures: First, the empty list is a list. Second, a complex term is a list if it consists of two items, the first of which is a term (called first), and the second of which is a list (called rest).
- `[mary|[john|[alex|[tom|[]]]]]`
- simpler notation: `[mary,john,alex,tom]`
- Exercise: Write a predicate `member/2`.

```
function D-RECOGNIZE (tape, machine) returns accept or reject
 index ← Beginning of tape
 current-state ← Initial state of machine
 loop
   if End of input has been reached then
     if current-state is an accept state then
       return accept
     else
       return reject
   elsif transition-table [current-state, tape[index]] is empty then
     return reject
  else
    current-state ← transition-table [current-state, tape[index]]
    index ← index + 1
end
```

function D-RECOGNIZE (*tape*, *machine*) **returns** accept or reject
  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
    if End of input has been reached **then**
      if current-state is an accept state **then**
        return accept
      else
        return reject
    elsif *transition-table [current-state, tape[index]]* is empty **then**
      return reject
    else
      current-state ← *transition-table [current-state, tape[index]]*
      index ← *index* + 1
end

```
% Finite state automaton.

fsa(Tape):-
initial(S),
fsa(Tape,S).

fsa([],S):- final(S).

fsa([H|T],S):-
trans_tab(S,H,NS),
fsa(T,NS).

% FSA transition table:
% trans_tab/3
% trans_tab(State, Input, New State)

trans_tab(1,a,1).
trans_tab(1,b,2).
trans_tab(2,a,2).

initial(1).
final(2).
```

Part VI

Context Free Grammars

Formal grammar

Definition

A formal grammar is a 4-tupel $G = (N, T, S, P)$ with

- *an alphabet of terminals T,*
- *an alphabet of nonterminals N with $N \cap T = \emptyset$,*
- *a start symbol $S \in N$,*
- *a finite set of rules/productions $P \subseteq \{\langle\alpha, \beta\rangle \mid \alpha, \beta \in (N \cup T)^* \text{ and } \alpha \notin T^*\}$.*

Instead of $\langle\alpha, \beta\rangle$ we write also $\alpha \to \beta$.

$$\begin{array}{lllllllll} S & \to & NP\ VP & VP & \to & V & NP & \to & D\ N \\ D & \to & \text{the} & N & \to & \text{cat} & V & \to & \text{sleeps} \end{array}$$

Generates: the cat sleeps

Formal grammar

Vocabulary

Let $G = (N, T, S, P)$ be a grammar and $v, w \in (T \cup N)^*$:

- v is directly derived from w (or w directly generates v), $w \rightarrow v$ if $w = w_1 \alpha w_2$ and $v = w_1 \beta w_2$ such that $\langle \alpha, \beta \rangle \in P$.
- v is derived from w (or w generates v), $w \rightarrow^* v$ if there exists $w_0, w_1, \ldots w_k \in (T \cup N)^*$ $(k \geq 0)$ such that $w = w_0$, $w_k = v$ and $w_{i-1} \rightarrow w_i$ for all $k \geq i \geq 0$.
- $\rightarrow^*$ denotes the reflexive transitive closure of $\rightarrow$
- $L(G) = \{w \in T^* | S \rightarrow^* w\}$ is the formal language generated by the grammar G.

S	→	NP VP	VP	→	V	NP	→	D N
D	→	the	N	→	cat	V	→	sleeps

Generates: the cat sleeps

Example

$$G_1 = \langle \{\text{S,NP,VP,N,V,D,N,EN}\}, \{\text{the, cat, peter, chases}\}, \text{S}, P \rangle$$

$$P = \left\{ \begin{array}{rcl rcl rcl} \text{S} & \rightarrow & \text{NP VP} & \text{VP} & \rightarrow & \text{V NP} & \text{NP} & \rightarrow & \text{D N} \\ \text{NP} & \rightarrow & \text{EN} & \text{D} & \rightarrow & \text{the} & \text{N} & \rightarrow & \text{cat} \\ \text{EN} & \rightarrow & \text{peter} & \text{V} & \rightarrow & \text{chases} & & & \end{array} \right\}$$

Example

$$G_1 = \langle \{\mathsf{S,NP,VP,N,V,D,N,EN}\}, \{\mathsf{the,\ cat,\ peter,\ chases}\}, \mathsf{S}, P \rangle$$

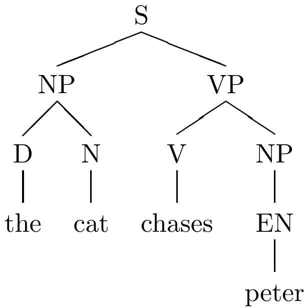
$$P = \left\{ \begin{array}{rcl rcl rcl} \mathsf{S} & \rightarrow & \mathsf{NP\ VP} & \mathsf{VP} & \rightarrow & \mathsf{V\ NP} & \mathsf{NP} & \rightarrow & \mathsf{D\ N} \\ \mathsf{NP} & \rightarrow & \mathsf{EN} & \mathsf{D} & \rightarrow & \mathsf{the} & \mathsf{N} & \rightarrow & \mathsf{cat} \\ \mathsf{EN} & \rightarrow & \mathsf{peter} & \mathsf{V} & \rightarrow & \mathsf{chases} & & & \end{array} \right\}$$

$$L(G_1) = \left\{ \begin{array}{cc} \text{the cat chases peter} & \text{peter chases the cat} \\ \text{peter chases peter} & \text{the cat chases the cat} \end{array} \right\}$$

"the cat chases peter" can be derived from S by:

$$\begin{array}{lll} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \rightarrow \mathsf{NP\ V\ NP} & \rightarrow \mathsf{NP\ V\ EN} \\ \rightarrow \mathsf{NP\ V\ peter} & \rightarrow \mathsf{NP\ chases\ peter} & \rightarrow \mathsf{D\ N\ chases\ peter} \\ \rightarrow \mathsf{D\ cat\ chases\ peter} & \rightarrow \mathsf{the\ cat\ chases\ peter} & \end{array}$$

Derivation tree



Chomsky-hierarchy

A grammar (N, T, S, P) is a

(right-linear) regular grammar (REG): iff every production is of the form $A \rightarrow \beta B$ or $A \rightarrow \beta$ with $A, B \in N$ and $\beta \in T^*$

context-free grammar (CFG): iff every production is of the form $A \rightarrow \beta$ with $A \in N$ and $\beta \in (N \cup T)^*$.

Chomsky-hierarchy

A grammar (N, T, S, P) is a

(right-linear) regular grammar (REG): iff every production is of the form
$A \rightarrow \beta B$ or $A \rightarrow \beta$ with $A, B \in N$ and $\beta \in T^*$

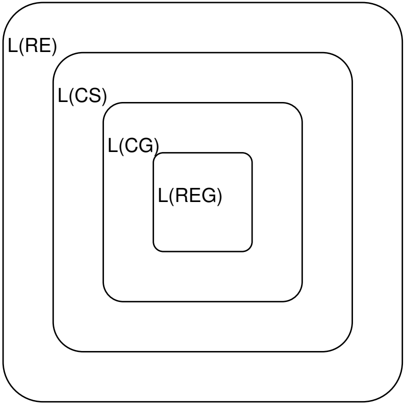
context-free grammar (CFG): iff every production is of the form $A \rightarrow \beta$ with $A \in N$ and $\beta \in (N \cup T)^*$.

context-sensitive grammar (CS): iff every production is of the form
$\gamma A \delta \rightarrow \gamma \beta \delta$ with $\gamma, \delta, \beta \in (N \cup T)^*, A \in N$ and $\beta \neq \epsilon$;
or of the form $S \rightarrow \epsilon$, in which case S does not occur on any right-hand side of a production.

recursively enumerable grammar (RE): if it is an arbitrary formal grammar.

Main theorem

$$L(REG) \subset L(CG) \subset L(CS) \subset L(RE)$$



regular languages

Definition

A grammar (N, T, S, P) *is a* *right-linear regular grammar* *iff all productions are of the form:*

$$A \longrightarrow w \text{ or } A \longrightarrow wB \text{ with } A, B \in N \text{ and } w \in T^*.$$

regular languages

Definition

A grammar (N, T, S, P) is a right-linear regular grammar iff all productions are of the form:

$$A \longrightarrow w \text{ or } A \longrightarrow wB \text{ with } A, B \in N \text{ and } w \in T^*.$$

Theorem

Every language generated by a right-linear regular grammar is a regular language and for every regular language there exists a right-linear regular grammar which generates it.

Exercise 9

Prove the proposition.

Proof: Each regular language is right-linear

$\Sigma = \{a_1, \ldots, a_n\}$

1. $\emptyset$ is generated by $(\{S\}, \Sigma, S, \{\})$,

Proof: Each regular language is right-linear

$\Sigma = \{a_1, \ldots, a_n\}$

1. $\emptyset$ is generated by $(\{S\}, \Sigma, S, \{\})$,
2. $\{\epsilon\}$ is generated by $(\{S\}, \Sigma, S, \{S \to \epsilon\})$,

Proof: Each regular language is right-linear

$\Sigma = \{a_1, \ldots, a_n\}$

1. $\emptyset$ is generated by $(\{S\}, \Sigma, S, \{\})$,
2. $\{\epsilon\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow \epsilon\})$,
3. $\{a_i\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow a_i\})$,

Proof: Each regular language is right-linear

$\Sigma = \{a_1, \ldots, a_n\}$

1. $\emptyset$ is generated by $(\{S\}, \Sigma, S, \{\})$,
2. $\{\epsilon\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow \epsilon\})$,
3. $\{a_i\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow a_i\})$,
4. If L_1, L_2 are regular languages with generating right-linear grammars (N_1, T_1, S_1, P_1), (N_2, T_2, S_2, P_2), then $L_1 \cup L_2$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S, P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$,

Proof: Each regular language is right-linear

$\Sigma = \{a_1, \dots, a_n\}$

1. $\emptyset$ is generated by $(\{S\}, \Sigma, S, \{\})$,
2. $\{\epsilon\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow \epsilon\})$,
3. $\{a_i\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow a_i\})$,
4. If L_1, L_2 are regular languages with generating right-linear grammars (N_1, T_1, S_1, P_1), (N_2, T_2, S_2, P_2), then $L_1 \cup L_2$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S, P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$,
5. $L_1 \circ L_2$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S_1, P_1' \cup_{\uplus} P_2)$ (P_1' is obtained from P_1 if all rules of the form $A \rightarrow w$ ($w \in T^*$) are replaced by $A \rightarrow wS_2$),

Proof: Each regular language is right-linear

$\Sigma = \{a_1, \dots, a_n\}$

1. $\emptyset$ is generated by $(\{S\}, \Sigma, S, \{\})$,
2. $\{\epsilon\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow \epsilon\})$,
3. $\{a_i\}$ is generated by $(\{S\}, \Sigma, S, \{S \rightarrow a_i\})$,
4. If L_1, L_2 are regular languages with generating right-linear grammars (N_1, T_1, S_1, P_1), (N_2, T_2, S_2, P_2), then $L_1 \cup L_2$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S, P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$,
5. $L_1 \circ L_2$ is generated by $(N_1 \uplus N_2, T_1 \cup T_2, S_1, P_1' \cup_{\uplus} P_2)$ (P_1' is obtained from P_1 if all rules of the form $A \rightarrow w$ ($w \in T^*$) are replaced by $A \rightarrow wS_2$),
6. L_1^* is generated by $(N_1, \Sigma, S_1, P_1' \cup \{S_1 \rightarrow \epsilon\})$ (P_1' is obtained from P_1 if all rules of the form $A \rightarrow w$ ($w \in T^*$) are replaced by $A \rightarrow wS_1$).

context-free grammars

Definition

A grammar (N, T, S, P) is context-free if all production rules are of the form:

$$A \rightarrow \alpha, \text{ with } A \in N \text{ and } \alpha \in (T \cup N)^*.$$

A language generated by a context-free grammar is said to be context-free.

context-free grammars

Definition

A grammar (N, T, S, P) is context-free if all production rules are of the form:

$$A \rightarrow \alpha, \text{ with } A \in N \textit{ and } \alpha \in (T \cup N)^*.$$

A language generated by a context-free grammar is said to be context-free.

Theorem

The set of context-free languages is a strict superset of the set of regular languages.

context-free grammars

Definition

A grammar (N, T, S, P) is context-free if all production rules are of the form:

$$A \rightarrow \alpha, \text{ with } A \in N \text{ and } \alpha \in (T \cup N)^*.$$

A language generated by a context-free grammar is said to be context-free.

Theorem

The set of context-free languages is a strict superset of the set of regular languages.

Proof: Each regular language is per definition context-free. $L(a^n b^n)$ is context-free but not regular ($S \rightarrow aSb, S \rightarrow \epsilon$).

Examples of context-free languages

- $L_1 = \{ww^R : w \in \{a, b\}^*\}$
- $L_2 = \{a^i b^j : i \geq j\}$
- $L_3 = \{w \in \{a, b\}^* : \text{more } a's \text{ than } b's\}$
- $L_4 = \{w \in \{a, b\}^* : \text{number of } a's \text{ equals number of } b's\}$

Examples of context-free languages

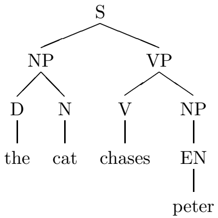
- $L_1 = \{ww^R : w \in \{a,b\}^*\}$
- $L_2 = \{a^i b^j : i \geq j\}$
- $L_3 = \{w \in \{a,b\}^* : \text{more } a's \text{ than } b's\}$
- $L_4 = \{w \in \{a,b\}^* : \text{number of } a's \text{ equals number of } b's\}$

$$\left\{\begin{array}{lllllllll} S & \rightarrow & aB & A & \rightarrow & a & B & \rightarrow & b \\ S & \rightarrow & bA & A & \rightarrow & aS & B & \rightarrow & bS \\ & & & A & \rightarrow & bAA & B & \rightarrow & aBB \end{array}\right\}$$

Derivation tree

$G_1 = \langle \{\mathsf{S,NP,VP,N,V,D,N,EN}\}, \{\mathsf{the,\ cat,\ peter,\ chases}\}, \mathsf{S}, P\rangle$

$$P = \left\{ \begin{array}{rcl rcl rcl} \mathsf{S} & \rightarrow & \mathsf{NP\ VP} & \mathsf{VP} & \rightarrow & \mathsf{V\ NP} & \mathsf{NP} & \rightarrow & \mathsf{D\ N} \\ \mathsf{NP} & \rightarrow & \mathsf{EN} & \mathsf{D} & \rightarrow & \mathsf{the} & \mathsf{N} & \rightarrow & \mathsf{cat} \\ \mathsf{EN} & \rightarrow & \mathsf{peter} & \mathsf{V} & \rightarrow & \mathsf{chases} & & & \end{array} \right\}$$



One derivation determines one derivation tree, but
the same derivation tree can result from different derivations.

Ambiguous grammars and ambiguous languages

Definition

Given a context-free grammar G: A derivation which always replaces the left furthest nonterminal symbol is called left-derivation

Ambiguous grammars and ambiguous languages

Definition

Given a context-free grammar G: A derivation which always replaces the left furthest nonterminal symbol is called left-derivation

Definition

A context-free grammar G is ambiguous iff there exists a $w \in L(G)$ with more than one left-derivation, $S \rightarrow^* w$.

Ambiguous grammars and ambiguous languages

Definition

Given a context-free grammar G: A derivation which always replaces the left furthest nonterminal symbol is called left-derivation

Definition

A context-free grammar G is ambiguous iff there exists a $w \in L(G)$ with more than one left-derivation, $S \rightarrow^* w$.

Definition

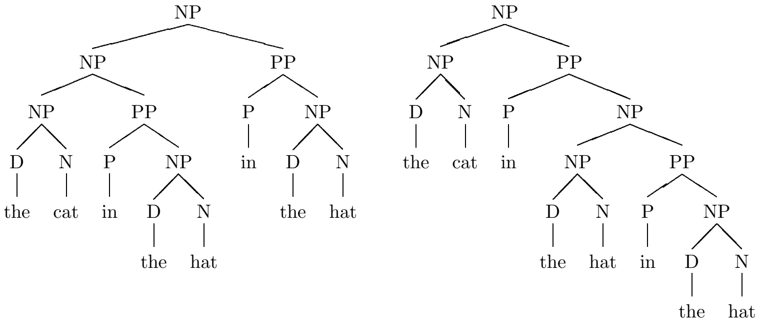
A context-free language L is ambiguous iff each context-free grammar G with $L(G) = L$ is ambiguous.

Left-derivations and derivation trees determine each other!

Example of an ambiguous grammar

$G = (N, T, \mathrm{NP}, P)$ with $N = \{\mathrm{D}, \mathrm{N}, \mathrm{P}, \mathrm{NP}, \mathrm{PP}\}$, $T = \{\mathrm{the}, \mathrm{cat}, \mathrm{hat}, \mathrm{in}\}$,

$$P = \left\{ \begin{array}{lllllllll} \mathrm{NP} & \rightarrow & \mathrm{D\ N} & \mathrm{D} & \rightarrow & \mathrm{the} & \mathrm{N} & \rightarrow & \mathrm{hat} \\ \mathrm{NP} & \rightarrow & \mathrm{NP\ PP} & \mathrm{N} & \rightarrow & \mathrm{cat} & \mathrm{P} & \rightarrow & \mathrm{in} \\ \mathrm{PP} & \rightarrow & \mathrm{P\ NP} & & & & & & \end{array} \right\}$$



Chomsky Normal Form

Definition

A grammar is in Chomsky Normal Form (CNF) if all production rules are of the form

1. $A \longrightarrow a$
2. $A \longrightarrow BC$

with $A, B, C \in T$ *and* $a \in \Sigma$ *(and if necessary* $S \rightarrow \epsilon$ *in which case* S *may not occur in any right-hand side of a rule).*

Theorem

Each context-free language is generated by a grammar in CNF.

Each context-free language is generated by a grammar in CNF

3 steps

1. Adapt the grammar such that terminals only occur in rules of type $A \longrightarrow a$.
2. Eliminate $A \longrightarrow B$ rules.
3. Eliminate $A \longrightarrow B_1 B_2 \dots B_n$ $(n > 2)$ rules.

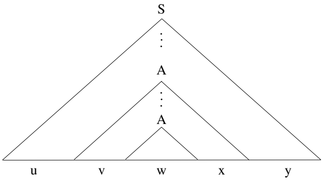
Pumping lemma for context-free languages

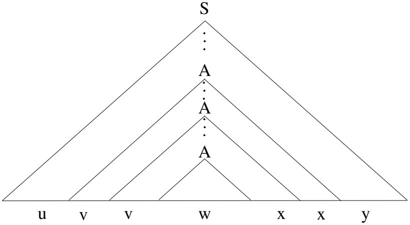
pumping lemma

For each context-free language L there exists a $p \in \mathbb{N}$ such that for any $z \in L$: if $|z| > p$, then z may be written as $z = uvwxy$ with

- $u, v, w, x, y \in T^*$,
- $|vwx| \leq p$,
- $vx \neq \epsilon$ and
- $uv^iwx^iy \in L$ for any $i \geq 0$.

Pumping lemma: proof sketch





$|vwx| \leq p$, $vx \neq \epsilon$ and $uv^iwx^iy \in L$ for any $i \geq 0$.

Existence of non context-free languages

- $L_1 = \{a^n b^n c^n\}$
- $L_2 = \{a^n b^m c^n d^m\}$
- $L_1 = \{ww : w \in \{a, b\}^*\}$

Closure properties of context-free languages

Theorem

Context-free languages are closed under

- *union*
- *concatenation*
- *Kleene's star*
- *intersection with a regular language*

union: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with
$P = P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$

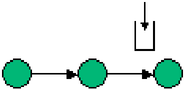
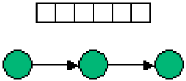
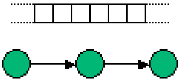
intersection: $L_1 = \{a^n b^n a^k\}$, $L_2 = \{a^n b^k a^k\}$, but $L_1 \cap L_2 = \{a^n b^n a^n\}$

complement: *de Morgan*

concatenation: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with
$P = P_1 \cup_{\uplus} P_2 \cup \{S \rightarrow S_1 S_2\}$

Kleene's star: $G = (N_1 \cup \{S\}, T_1, S, P)$ with $P = P_1 \cup \{S \rightarrow S_1 S, S \rightarrow \epsilon\}$

Chomsky-hierarchy (1956)

Type 3: REG	finite-state automaton		WP: linear
Type 2: CF	pushdown-automaton		WP: cubic
Type 1: CS	linearly restricted automaton		WP: exponential
Type 0: RE	Turing machine		WP: not decid-able

Part VII

Parsing

example grammar

'syntactical rules'

$S \longrightarrow NP\ VP$

$VP \longrightarrow V\ NP$

$VP \longrightarrow VP\ PP$

$NP \longrightarrow NP\ PP$

$PP \longrightarrow P\ NP$

'lexical rules'

NP → John

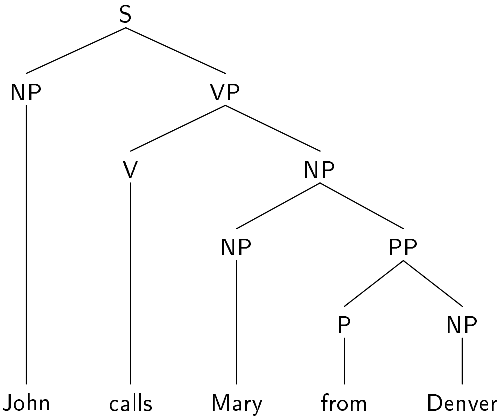
NP → Mary

NP → Denver

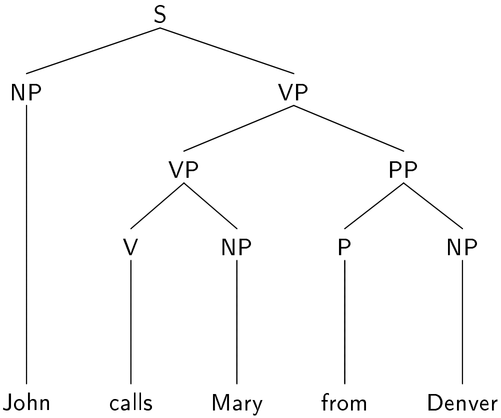
V → calls

P → from

derivation tree



derivation tree



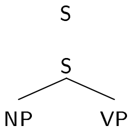
top-down search

John calls Mary from Denver

S

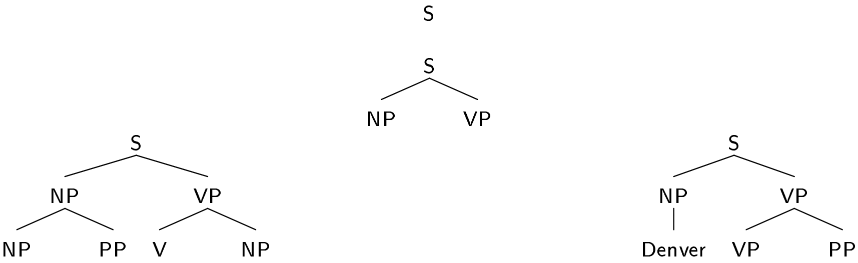
top-down search

John calls Mary from Denver



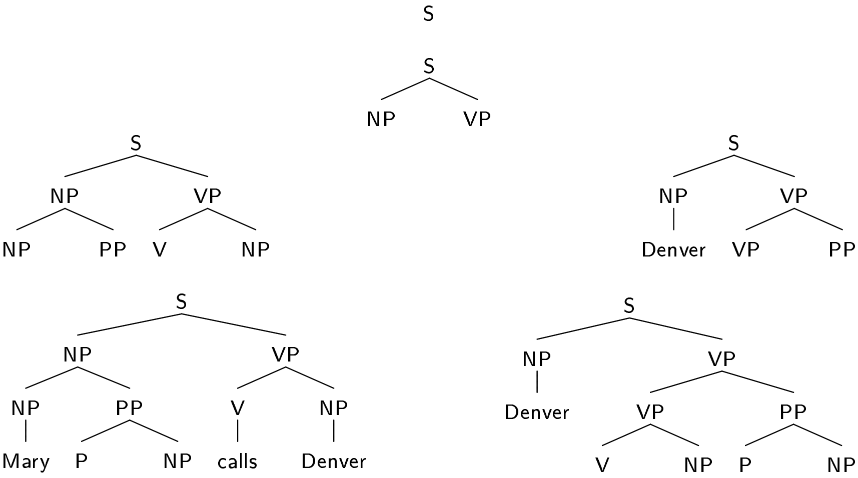
top-down search

John calls Mary from Denver



top-down search

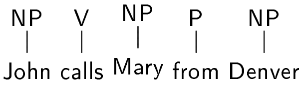
John calls Mary from Denver



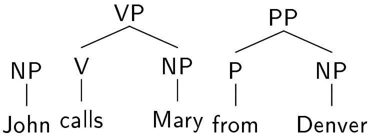
bottom-up search

John calls Mary from Denver

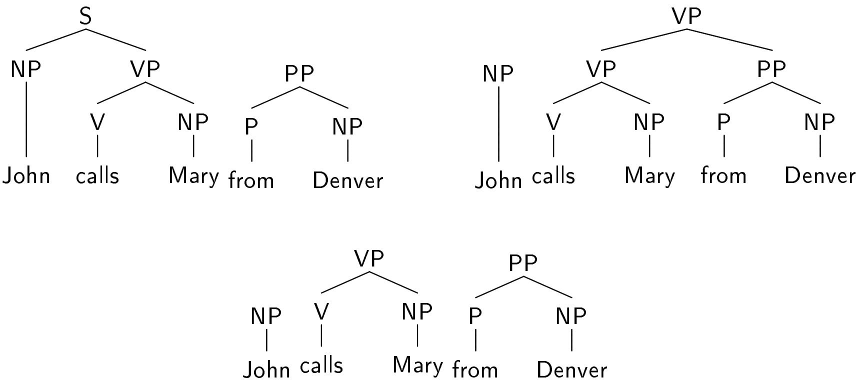
bottom-up search



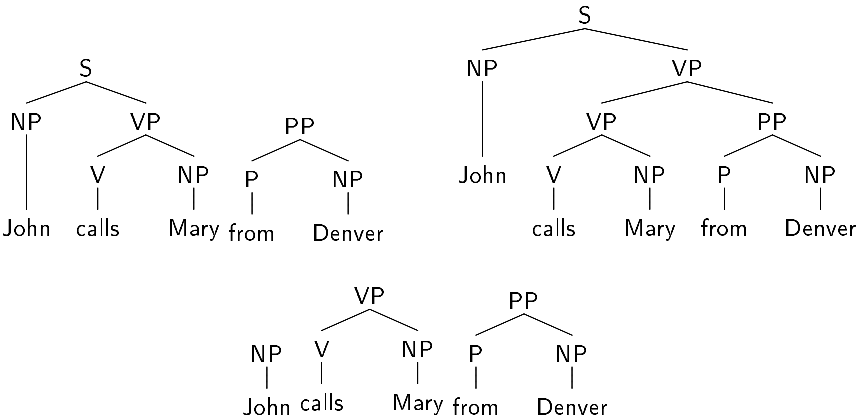
bottom-up search



bottom-up search



bottom-up search



search strategies

- top-down
- bottom-up

- depth-first
- breadth-first

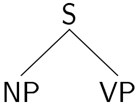
- left-to-right
- right-to-left

Example: top-down, depth-first, left-to-right parse

S

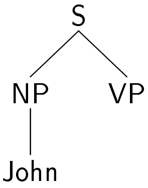
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



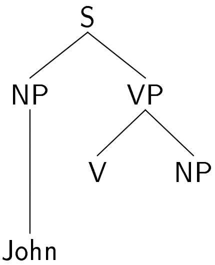
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



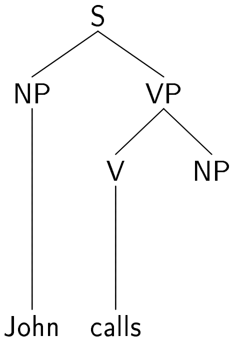
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



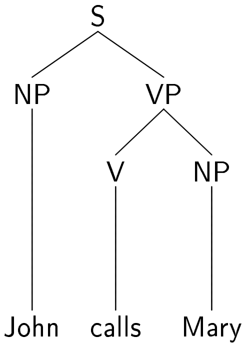
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



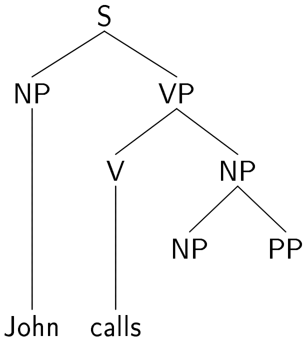
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



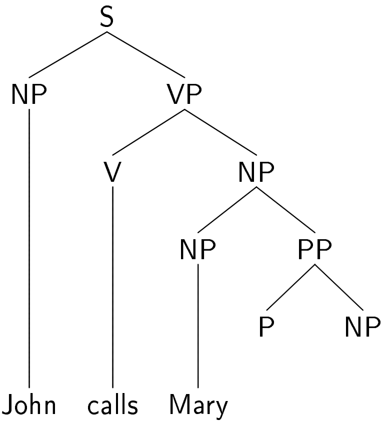
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



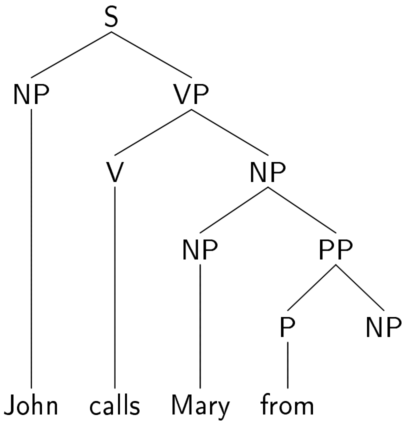
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



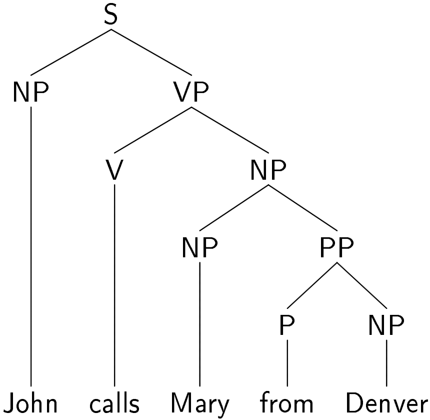
John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



John calls Mary from Denver

Example: top-down, depth-first, left-to-right parse



John calls Mary from Denver

left-recursion is dangerous for top-down, left-to-right

additional rules:

$NP \longrightarrow D\ N$

$D \longrightarrow a$

$N \longrightarrow friend$

Parse "a friend calls Mary from Denver"

empty expansions are dangerous for bottom-up

additional rules:

$NP \longrightarrow D\ N$

$D \longrightarrow a$

$D \longrightarrow \epsilon$

$N \longrightarrow friend$

$N \longrightarrow friends$

Parse “friends call Mary from Denver”

problems with simple parsing strategies

- top-down: left-recursions
- bottom-up: empty expansions
- lots of avoidable redoes (example: parse “flights from Düsseldorf to Riga by Airbaltic” top-down as an NP)
- ambiguities (Example: Show me the meal on the flight from Düsseldorf to Riga by Airbaltic)

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John

calls

Mary

from

Denver

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**				
calls		**V**			
Mary			**NP**		
from				**P**	
Denver					**NP**

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**	–			
calls		**V**	*VP*		
Mary			**NP**	–	
from				**P**	*PP*
Denver					**NP**

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**	–	*S*		
calls		**V**	*VP*		
Mary			**NP**	–	
from				**P**	*PP*
Denver					**NP**

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**	–	*S*		
calls		**V**	*VP*	–	
Mary			**NP**	–	
from				**P**	*PP*
Denver					**NP**

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**	–	*S*		
calls		**V**	*VP*	–	
Mary			**NP**	–	*NP*
from				**P**	*PP*
Denver					**NP**

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**	–	*S*	–	
calls		**V**	*VP*	–	
Mary			**NP**	–	*NP*
from				**P**	*PP*
Denver					**NP**

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**	–	*S*	–	
calls		**V**	*VP*	–	VP_1, VP_2
Mary			**NP**	–	*NP*
from				**P**	*PP*
Denver					**NP**

CYK-parser (Cocke-Kasami-Younger)

precondition: CFG grammar in CNF

John	**NP**	–	S	–	S_1, S_2
calls		**V**	VP	–	VP_1, VP_2
Mary			**NP**	–	NP
from				**P**	PP
Denver					**NP**

exercises overview

- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Exercise 6
- Exercise 7
- Exercise 8
- Exercise 9