Course 1
Introduction to Automata Theory

The structure and the content of the lecture is based on http://www.eecs.wsu.edu/~ananth/CptS317/Lectures/index.htm

Major Notions

1. Automata Theory and Formal Languages
2. Context-Free Grammars
3. Turing Machines

Why Study Automata Theory and Formal Languages?

- Regular expressions (REs) are used in many systems.
  - E.g., UNIX, Linux, OS X,… a.*b.
  - E.g., Document Type Definitions describe XML tags with a RE format like person (name, addr, child*).

Why Study Automata Theory and Formal Languages? (cont'd)

- Finite automata model protocols, electronic circuits.
  - Theory is used in *model-checking*.

Why Context-Free Grammars?

- Context-free grammars (CFGs) are used to describe the syntax of essentially every modern programming language.
- Every modern complier uses CFG concepts to parse programs.
  - Role in describing natural languages.
- Document Type Definitions are CFG's.

Why Turing Machines?

- When developing solutions to real problems, we encounter situations that software can not do.
  - Undecidable things – no program can do it 100% of the time with 100% accuracy.
  - Intractable things – there are programs, but no fast programs.

What is Automata Theory?

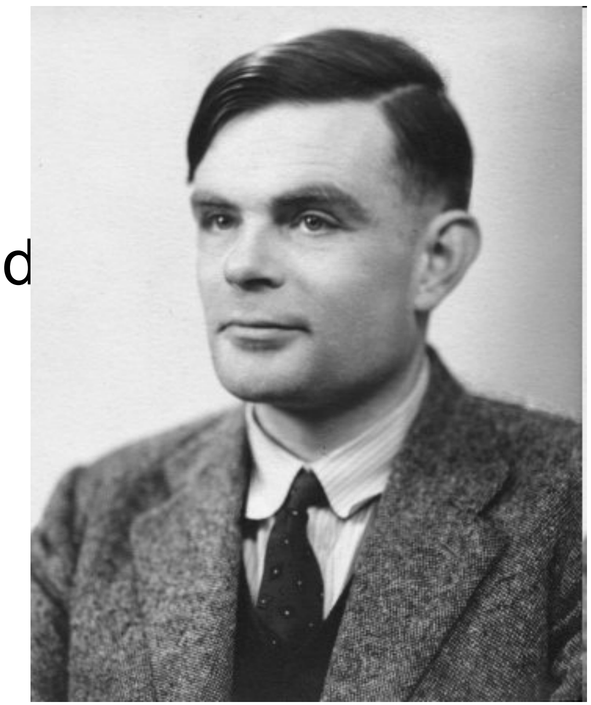
- *Study of abstract computing devices, or "machines"*
- Automaton = an abstract computing device
  - Note: A "device" need not even be a physical hardware!
- A fundamental question in computer science:
  - Find out what different models of machines can do and cannot do
  - The *theory of computation*
- Computability vs. Complexity

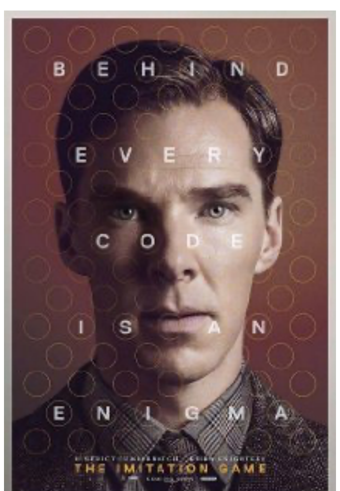
(A pioneer of automata theory)

Alan Turing (1912-1954)

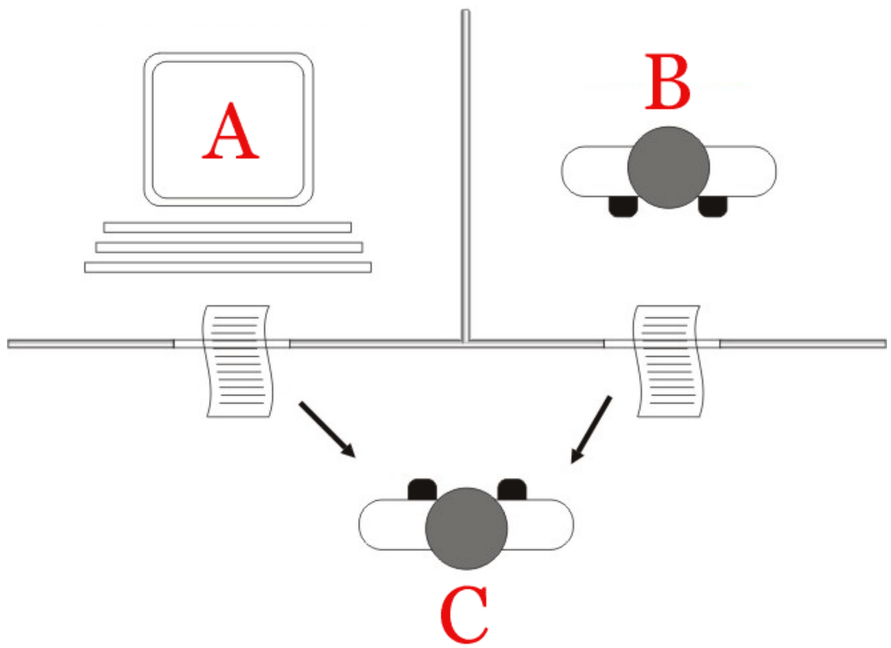
- Father of Modern Computer Science
- English mathematician
- Studied abstract machines called ***Turing machines*** even before computers existed

Heard of the Turing test?





Turing Test



https://en.wikipedia.org

FYI: Ex Machina (movie)

Theory of Computation: A Historical Perspective

1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• "Finite automata" machines studied • Noam Chomsky proposes the "Chomsky Hierarchy" for formal languages
1969	Cook introduces "intractable" problems or "NP-Hard" problems
1970-	Modern computer science: compilers, computational & complexity theory evolve

Languages & Grammars

Or "**words**"

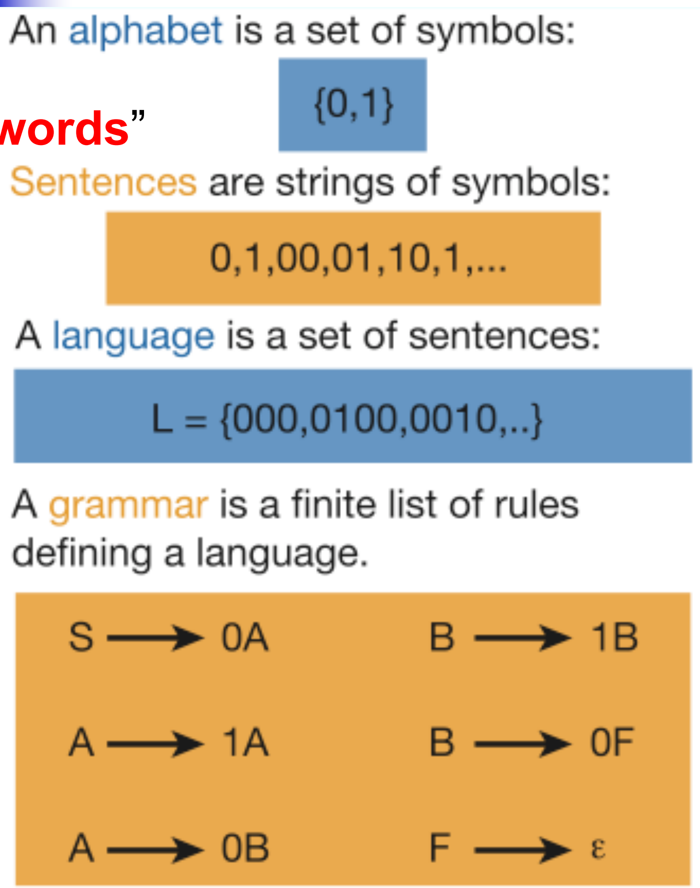
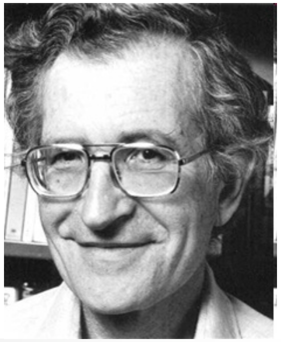


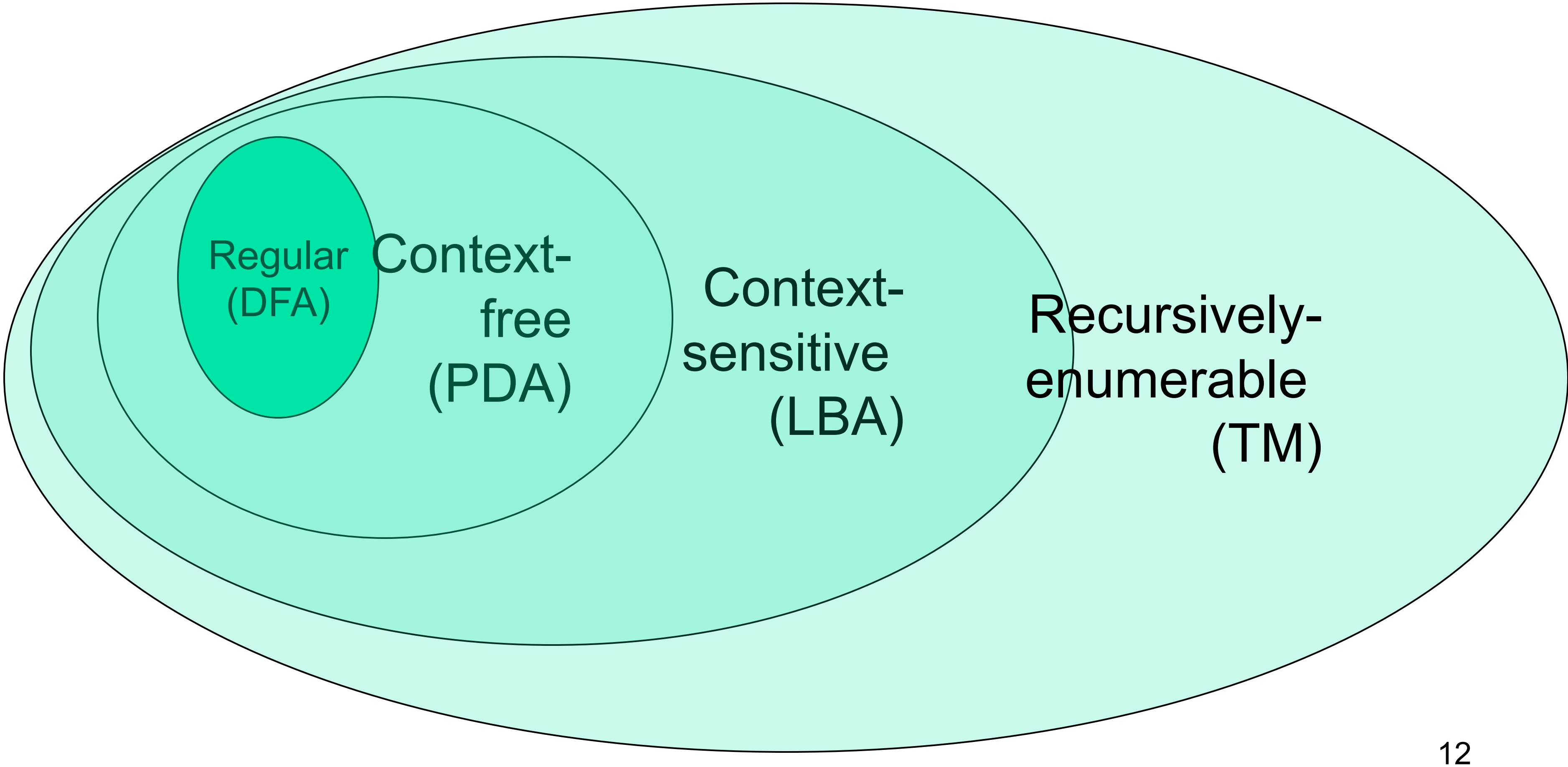
Image source: Nowak et al. Nature, vol 417, 2002

- <u>Languages</u>: "*A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"
- <u>Grammars</u>: "*A grammar can be regarded as a device that enumerates the sentences of a language*" - nothing more, nothing less
- *N. Chomsky, Information and Control, Vol 2, 1959*

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



The Central Concepts of Automata Theory

Alphabet

An alphabet is a finite, non-empty set of symbols

- We use the symbol ∑ (sigma) to denote an alphabet
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}
  - …

Strings

A string or word is a finite sequence of symbols chosen from ∑

- ***Empty string is ε (or "epsilon")***

- Length of a string *w,* denoted by "|*w*|", is equal to the *number of (non- ε) characters in the string*
  - *E.g., x = 010100* |*x*| = *6*
  - *x = 01 ε 0 ε 1 ε 00 ε* |*x*| = *?*

- *xy = concatenation* of two strings *x* and *y*

Powers of an alphabet

Let ∑ be an alphabet.

- Σ^k = the set of all strings of length k
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

Languages

*L is a said to be a language over alphabet ∑, only if L ⊆ ∑**

➔ this is because ∑* is the set of all strings (of all possible length including 0) over the given alphabet ∑

<u>Examples:</u>

1. Let L be *the* language of <u>all strings consisting of *n* 0's followed by *n* 1's</u>:

   L = {ε, 01, 0011, 000111,…}

2. Let L be *the* language of <u>all strings of with equal number of 0's and 1's</u>:

   L = {ε, 01, 10, 0011, 1100, 0101, 1010, 1001,…}

   → Canonical ordering of strings in the language

Definition: Ø denotes the Empty language

- Let L = {ε}; Is L=Ø? **NO**

The Membership Problem

Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$.

<u>Example:</u>

Let w = 100011

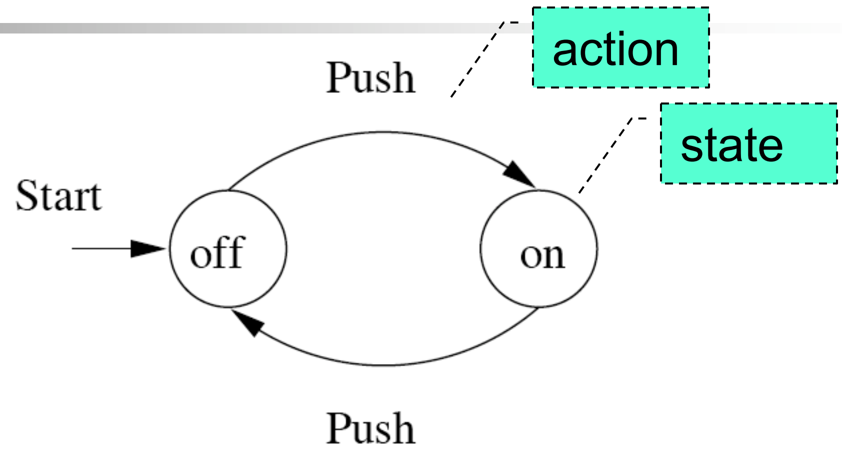
Q) Is $w \in$ the language of strings with equal number of 0s and 1s?

Finite Automata

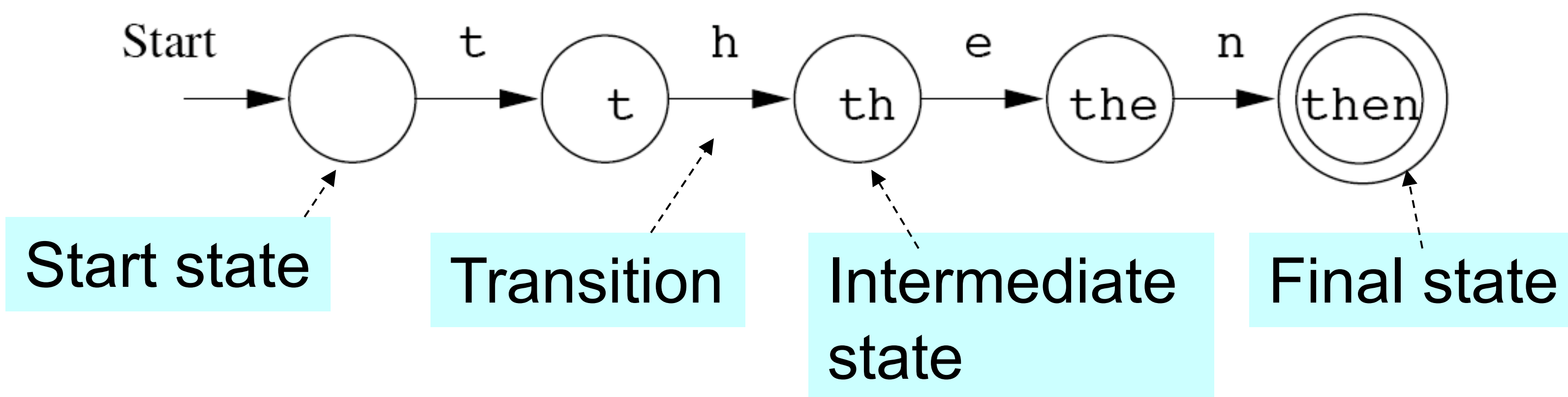
- Some Applications
  - Software for designing and checking the behavior of digital circuits
  - Lexical analyzer of a typical compiler
  - Software for scanning large bodies of text (e.g., web pages) for pattern finding
  - Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol)

Finite Automata : Examples

- On/Off switch

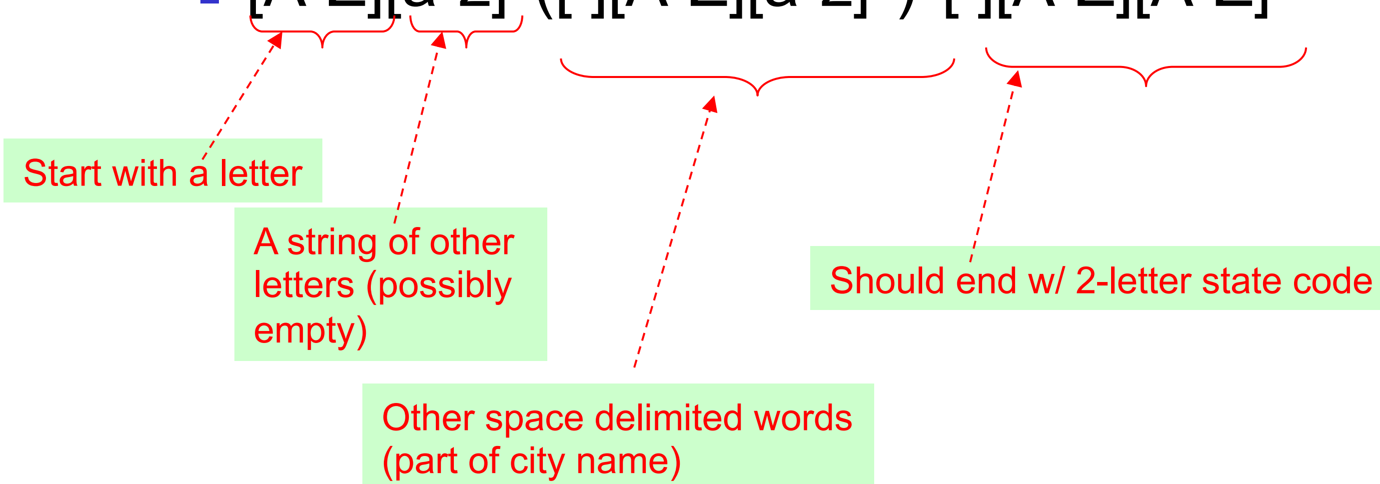


- Modeling recognition of the word "*then*"



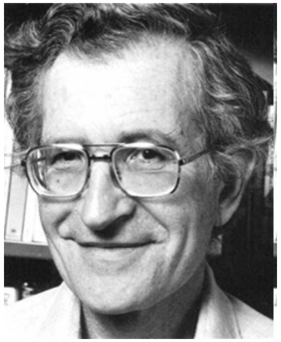
Structural expressions

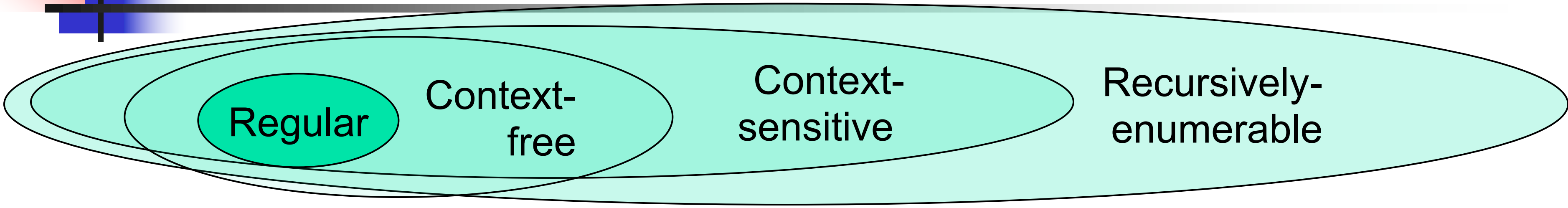
- Grammars
- Regular expressions
  - E.g., unix style to capture city names such as “Palo Alto CA”:
    - [A-Z][a-z]*([ ][A-Z][a-z]*)*[ ][A-Z][A-Z]



The Chomsky Hierarchy

The Chomsky Hierarchy





Grammar	Languages		Automaton	Production Rules
Type-0	Recursively enumerable	$\mathcal{L}_0$	Turing machine	$\alpha \rightarrow \beta$
Type-1	Context sensitive	$\mathcal{L}_1$	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	$\mathcal{L}_2$	Non-deterministic push down automaton	$A \rightarrow \gamma$
Type-3	Regular	$\mathcal{L}_3$	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$

The Chomsky Hierarchy (cont'd)

Classification using the structure of their rules:

- *Type-0 grammars*: there are no restriction on the rules;

- *Type-1 grammars/Context sensitive grammars*: the rules for this type have the next form:

$$uAv \to upv,\ u, p, v \in V_G^*,\ p \neq \lambda,\ \mathrm{A} \in V_N$$

or $A \to \lambda$ and in this case A does not belong to any right side of a rule.

Remark. The rules of the second form have sense only if A is the start symbol.

The Chomsky Hierarchy (cont'd)

Remarks

1. A grammar is *Type 1 monotonic* if it contains no rules in which the left-hand side consists of more symbols than the right-hand side. This forbids, for instance, the rule , . $NE \rightarrow and\ N$, where N, E are non-term. symb.; and is a terminal symb ($3 = |.NE| \geq |and\ N| = 2$).

The Chomsky Hierarchy (cont'd)

Remarks

- A grammar is *Type 1 context-sensitive* if all of its rules are context-sensitive. A rule is context-sensitive if actually only one (non-terminal) symbol in its left-hand side gets replaced by other symbols, while we find the others back undamaged and in the same order in the right-hand side.
- **Example**: $Name\ Comma\ Name\ End \rightarrow Name\ and\ Name\ End$ meaning that the rule $Comma \rightarrow and$ may be applied if the left context is $Name$ and the right context is $Name\ End$. The contexts themselves are not affected. The replacement must be at least one symbol long; this means that context-sensitive grammars are always monotonic.

The Chomsky Hierarchy (cont'd)

Classification using the structure of their rules:

- *Type-2 grammars/Context free grammars*: the rules for this type are of the form:

$$A \rightarrow p,\ p \in V_G^*,\ \mathrm{A} \in V_N$$

- *Type-3 grammars/*regular grammars: the rules for this type have one of the next two forms:

Cat. I rules	$A \rightarrow Bp$	or	$A \rightarrow pB$
Cat. II rules	$\mathrm{C} \rightarrow q$		$\mathrm{C} \rightarrow q$

$$\mathrm{A}, B, C \in V_N,\ p, q \in V_T^*$$

- Rule $A \rightarrow \lambda$ is allowed if A does not belongs to any right side of a rule.

Summary

- Automata theory & a historical perspective
- Chomsky hierarchy
- Finite automata
- Alphabets, strings/words/sentences, languages
- Membership problem
- Chomsky hierarchy