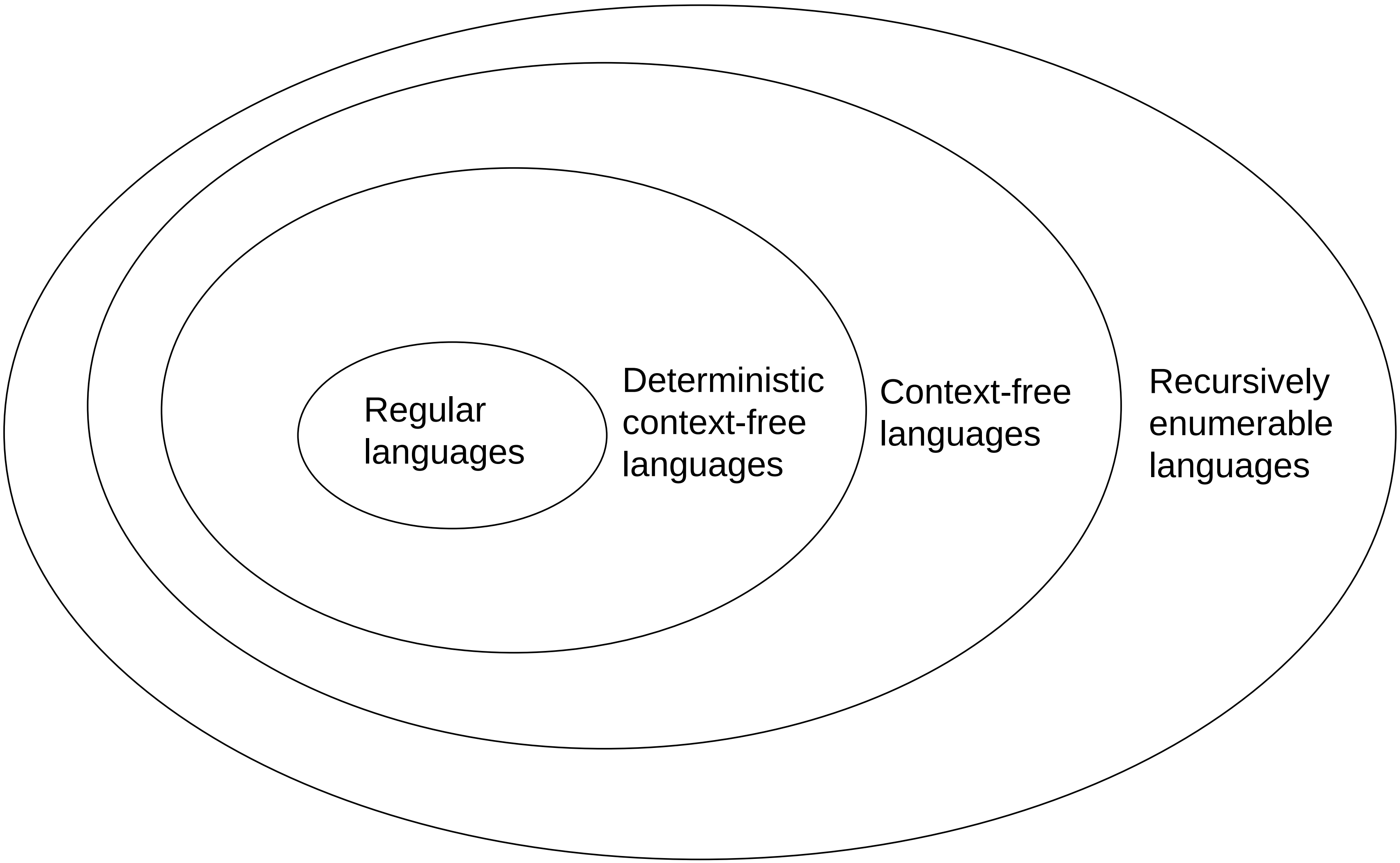
Theory of Computation

Regular Expressions

Lecture 10a - Manuel Mazzara

Recap



Regular languages
Deterministic context-free languages
Context-free languages
Recursively enumerable languages

Regular Expressions

- Regular expressions and finite-state automata represent **regular languages**
- The basic regular expression operations are: **concatenation**, **union**, and **Kleene closure**
- The regular expression language is a powerful **pattern-matching tool**
- Any regular expression can be converted into a FSA

Classes of models

- Languages can be represented through
  - Sets
  - Patterns
  - Regular expressions
  - Operational models
    - Automata
    - Petri Nets
    - Statecharts
    - Transducers
  - Generative models
    - Grammars
  - Declarative models
    - Logic

Recognizing languages

- <u>Automata</u> are a means to recognize languages
- An automaton describes the system behavior through <u>states</u>, <u>transitions</u> among states (and sometimes the use of <u>memory</u>)
- Different automata (or machines) lead to different <u>expressive powers</u>
  - Finite state automata
  - Pushdown automata
  - Turing Machine
  - Other models (nondeterminism…)

Generating languages

- A language can be described through the rules that generate all the strings of the language
  - The rules MUST be able to generate all the strings
  - The strings that do not belong to the language MUST NOT be generated
- Grammars are formed by sets of rules to describe languages
  - Limiting the way to express rules limits the expressive power of the generated language

Regular Expressions

- Inductive definition of REs over an alphabet Σ:
  - $\emptyset$ is a regular expression (denoting the language $\emptyset$)
  - The empty string is a RE (denoting the language $\{\varepsilon\}$)
  - Each symbol of Σ is a RE (denoting $\{a\}$, $a \in \Sigma$)
  - Let r and s be two REs, then:
    - (r.s) is a RE (denoting r concatenated with s)
      - For simplicity, the dot is often omitted
    - (r | s) is a RE (denoting r union s)
      - Sometimes written r + s or $r \cup s$
    - (r)* is a RE (denoting the smallest superset of r containing ε and closed under .)
  - Nothing else is a RE

Example

((0.(0|1)*)|((0|1)*.0))

- It is a regular expression over the alphabet {0,1}
  - Strings that **start with 0**
  - Strings that **end with 0**

REs and regular grammars

- **REs exactly correspond to regular languages** (same power as FSAs and RGs – will see later)
- Proof:
  - Every language denoted by a RE is regular:
    - Look at the inductive definition
    - We know how to build FSA for base cases
    - Regular languages are closed under concatenation, *, union and we know hot to build the FSA

REs in practice

- REs are very common in practice:
  - **Lexical analyzers** for artificial languages (ex. lex)
  - Advanced find&replace **features in text editors** and system tools (emacs, vi, grep...)
  - Scripting languages such as Perl, Python, Ruby, Scheme...
- There is a IEEE POSIX standard (standard API for unix/linux) also for REs
- Syntactically, REs in "practice" are a little different from what was shown in the previous slides...

POSIX REs

- Metacharacters: ( ) . [ ] ^ \ $ * + ? | { }
- Warning: . Is used to indicate any character, not to concatenate!
- [α] denotes a single character $\in \alpha$ (ex. [abc] = {a,b,c}. One can also write [a-z] to indicate any lower case letter)
- [^α]: negation: any symbol $\notin \alpha$ (ex. [^a-z] is any character that is not a lower case letter)

POSIX REs (cont.)

- ^ and $ denote ε at the beginning and, respectively, at the end of a line of text
- *,+,|,(,) are as usual
- \ serves as "escape" (for example, \$ denotes the $ character)

POSIX REs – Example 1

- `^[hc]at` matches "hat" and "cat", but only at the beginning of the string or line

- `[hc]at$` matches "hat" and "cat", but only at the end of the string or line

- `?` Matches the preceding element zero or one time
    - `ab?c` matches only "ac" or "abc"

POSIX REs – Example 2

>> grep -E "^((Two)|(In))" grep.txt

In addition, two variant programs egrep and fgrep are available. Egrep is

Two regular expressions may be concatenated; the resulting regular

Two regular expressions may be joined by the infix operator |; the

In basic regular expressions the metacharacters ?, +, {, |, (, and) lose

In egrep the metacharacter { loses its special meaning; instead use \{ .

<u>**All lines starting with “Two” or “In”**</u>

POSIX REs – Example 3

>> grep -E "^[A-Z]+$" grep.txt

SYNOPSIS

DESCRIPTION

DIAGNOSTICS

BUGS

<u>All lines consisting of upper case letters</u>

POSIX REs – Example 4

>> grep -E "^[^a-zA-Z].+\.$" grep.txt

-S Search subdirectories.

[:print:], [:punct:], [:space:], [:upper:], and [:xdigit:].

[[:alnum:]] and \W is a synonym for [^[:alnum]].

? The preceding item is optional and matched at most once.

** The preceding item will be matched zero or more times.*

+ The preceding item will be matched one or more times.

{ n } The preceding item is matched exactly n times.

All lines starting with a non-letter character and ending with a dot

More operators

- $\alpha?$ — α is optional
- $\alpha\{n\}$ — α^n
- $\alpha\{n,m\}$ — $\alpha^n \cup \alpha^{n+1} \cup \alpha^{n+2} \cup .. \cup \alpha^m$

Example 5

>> grep -E "[a-zA-Z]{15}" grep.txt

grep [-[[AB]]<num>] [-[CEFGLSVbchilnqsvwx?]] [-[ef]] <expr> [<files...>]

available functionality using either syntax. In other implementations,

<u>All lines containing a string of at least 15 letters</u>

Regular Expressions - recap

- Regular expressions and finite-state automata represent **regular languages**
- The basic regular expression operations are: **concatenation**, **union**, and **Kleene closure**
- The regular expression language is a powerful **pattern-matching tool**
- Any regular expression can be converted into a (N)FSA

Regular Expression to NFSA

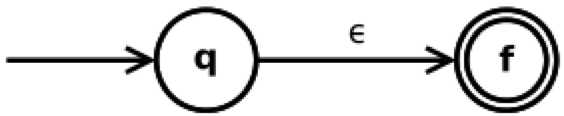
- **Thompson's construction** is an algorithm for transforming a regular expression into an equivalent nondeterministic finite state automaton

- We present here only a **sketch of the construction** omitting details (from Aho et al., 1986)

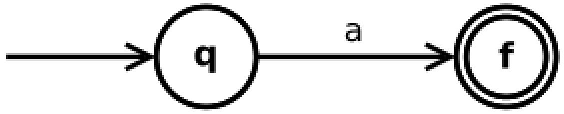
- Online tool:
  - http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

Construction Rules (1)

The **empty-expression** ε is converted to

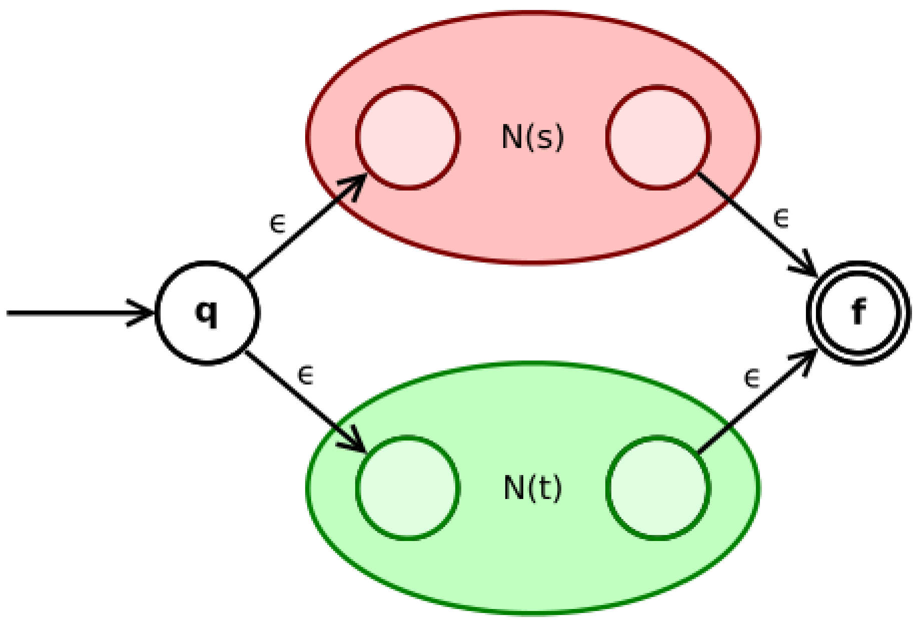


A **symbol** *a* of the input alphabet is converted to



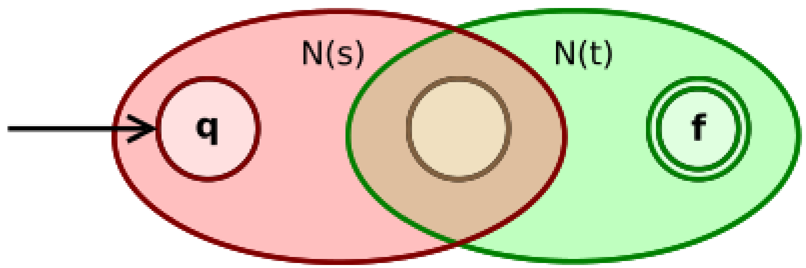
Construction Rules (2)

The **union expression** ***s|t*** is converted to



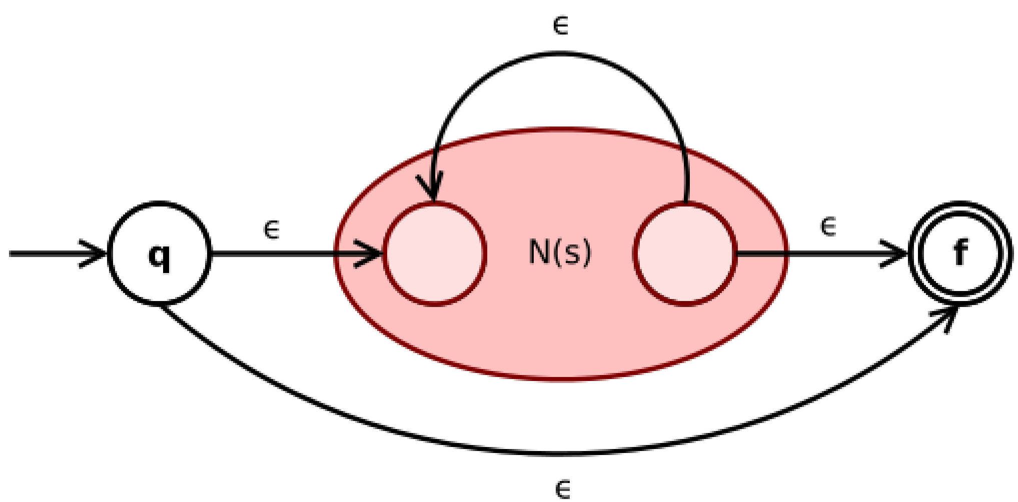
Construction Rules (3)

The **concatenation expression** *st* is converted to

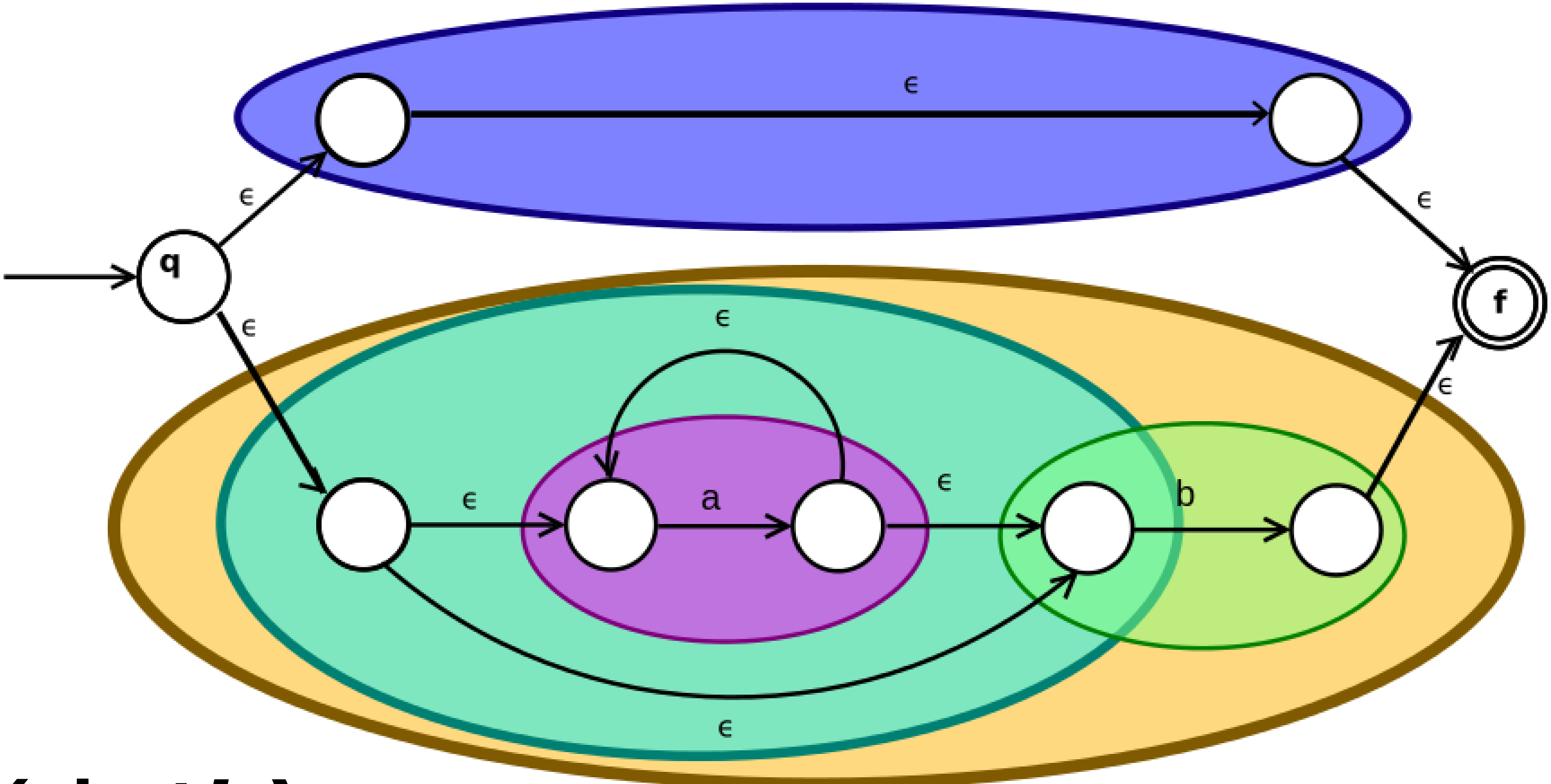


Construction Rules (4)

The **Kleene star expression** s^* is converted to



Example



(ε|a*b)

Equivalence

- Thompson's construction is one of several algorithms for constructing NFAs from regular expressions
- **Kleene's algorithm**
  - transforms given deterministic finite automaton into a regular expression (<u>not presented here</u>)
- Thompson and Kleene algorithms plus several others establish the equivalence of description formats for **regular languages**

Theory of Computation

Generative Grammars

Lecture 10b - Manuel Mazzara

Models for languages

Models suitable to recognize/accept, translate, compute languages

- They “receive” an input string and process it

→**Operational models (Automata)**

Models suitable to describe how to generate a language

- Sets of rules to build phrases of a language

→**Generative models (Grammars)**

Grammars (1)

- **Generative models** produce strings
  - grammar (or syntax)
- **A grammar is a set of rules** to build the phrases of a language
  - It applies to any notion of language
- **A formal grammar** generates strings of a language through a rewriting process

Rewriting

- Rewriting relevant to many fields
  - Mathematics
  - Computer science
  - Logic
- It consists of a wide range of methods for **replacing subterms** of a “formula” with other terms
  - Potentially nondeterministic

Examples

- Semantically equivalent formulae in propositional logic
  - $A \wedge B$ can be replaced with $\sim(\sim A \vee \sim B)$
  - $\sim A \vee B$ can be replaced with $A \Rightarrow B$
  - …
- Examples of tautologies in FOL
  - We can rewrite the tautology $\sim A \vee A$ by replacing A with a w.f.f. of propositional or FOL logic

Linguistic rules (1)

- **Natural languages** are explained through rules such as:
  - A phrase is made of a **subject followed by a predicate**
  - A subject can be a noun or a pronoun or…
  - A predicate can be a verb followed by a complement
- Programming languages are expressed similarly:
  - A program consists of a **declarative part** and an **executable part**
  - The declarative part …
  - The executable part consists of a statement sequence
  - A statement can be …

Linguistic rules (2)

- In general, a **linguistic rule** describes a “main object”
  – Examples: a book, a program, a message, …

  as a sequence of “composing objects”
- Each “composing object” is “refined” by replacing it with more detailed objects and so on… until a sequence of base elements is obtained

Grammars (2)

- **A grammar is a linguistic rule**
- It is composed by
  - a main object: **initial symbol**
  - composing objects: **nonterminal symbols**
  - base elements: **terminal symbols**
  - refinement rules: **productions**
- Formally?

Noam Chomsky (1)

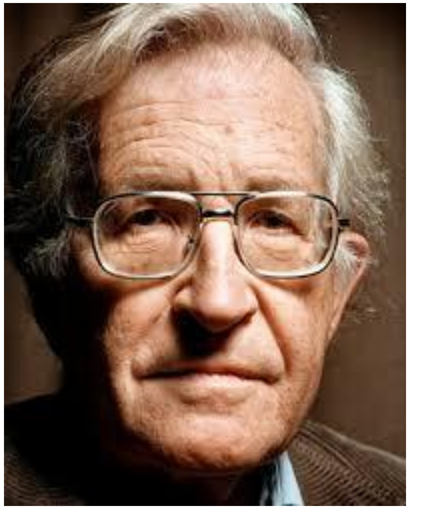
"A grammar can be regarded as **a device that enumerates the sentences** of a language"

"A grammar of *L* can be regarded as a function whose range is exactly *L*"

Noam Chomsky

On Certain Formal Properties of Grammars

Information and Control, Vol 2, 1959

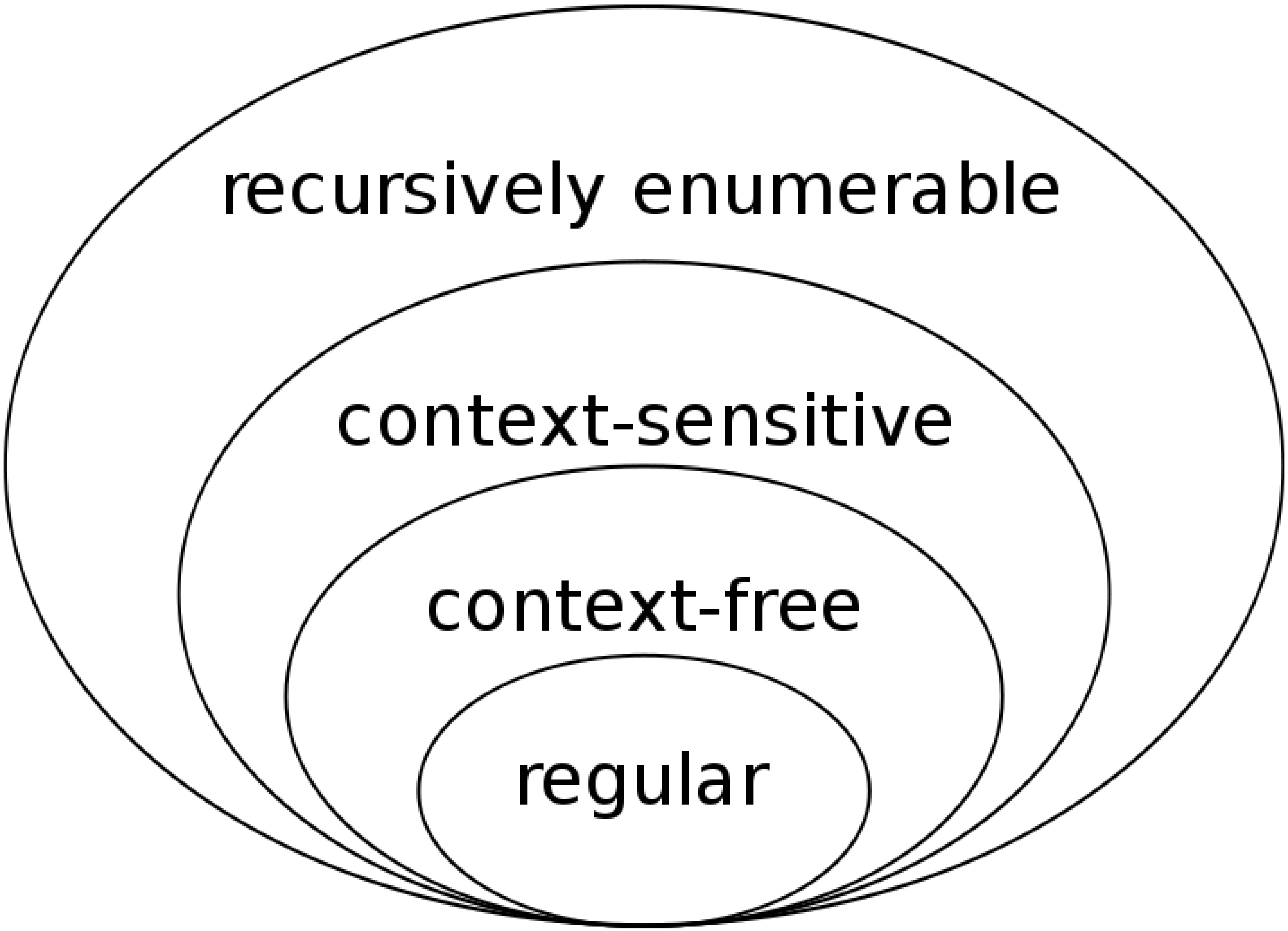


Noam Chomsky (2)

Avram Noam Chomsky (born December 7, 1928) is an American linguist, philosopher, cognitive scientist, historian, logician, social critic, and political activist. – **Wikipedia**

The “father of modern linguistics”

Chomsky hierarchy



Definition

- A grammar is a tuple <V_N, V_T, P, S> where
  - V_N is the **nonterminal alphabet** (or vocabulary)
  - V_T is the **terminal alphabet** (or vocabulary)
  - $V = V_N \cup V_T$
  - $S \in V_N$ is a particular element of V_N called axiom or **initial symbol**
  - $P \subseteq V^* \cdot V_N \cdot V^* \times V^*$ is the (finite) set of rewriting rules or **productions**
- A grammar G=<V_N, V_T, P, S> generates a language on the alphabet V_T

Productions

- A production is an element of $V^* \cdot V_N \cdot V^* \times V^*$
  - This is usually denoted as $<\alpha, \beta>$, where $\alpha \in V^* \cdot V_N \cdot V^*$ and $\beta \in V^*$
- We generally indicate a production as $\alpha \rightarrow \beta$
  - **α is a sequence of symbols including at least one nonterminal symbol**
  - **β is a (potentially empty) sequence of (terminal or non terminal) symbols**

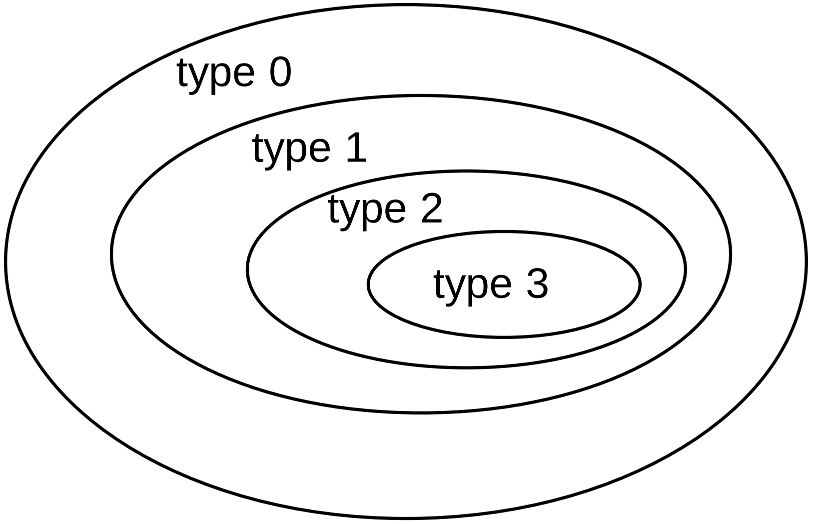
Example

- $V_N = \{S, A, B, C, D\}$
- $V_T = \{a,b,c\}$
- S is the initial symbol
  - It is not mandatory to call it S
- $P = \{ \quad S \rightarrow AB,$
  $BA \rightarrow cCD,$
  $CBS \rightarrow ab,$
  $A \rightarrow \varepsilon\}$

$\rightarrow$ The generated language is on the alphabet $\{a,b,c\}$

Chomsky hierarchy (1)

- Grammars are classified according to **the form of their productions**
- Chomsky classified grammars in four types



Chomsky hierarchy (2)

- **<u>Type 3 grammars</u>** restrict productions to **a single nonterminal on the left-hand side** and a **right-hand side consisting of a single terminal**, possibly followed (or preceded, but not both in the same grammar) by a single nonterminal
  - The rule $S \rightarrow \varepsilon$ is also allowed here if *S* does not appear on the right side of any rule
- **<u>Type-2 grammars</u>** are defined by rules of the form $A \rightarrow \gamma$ where **A is a nonterminal** and γ **is a string of terminals and nonterminals**

Chomsky hierarchy (3)

- **Type-1 grammars** have rules of the form $\alpha A\beta \rightarrow \alpha\,\gamma\,\beta$, where A is a nonterminal and α, β and γ are strings of terminals and nonterminals.
  - γ must be non-empty
  - The rule $S \rightarrow \varepsilon$ is allowed if *S* does not appear on the right side of any rule
- **Type-0 grammars** include all formal grammars

Immediate derivation relation

$\alpha \Rightarrow \beta$ (β is obtained by immediate derivation from α)

- $\alpha \in V^{*} \cdot V_N \cdot V^{*}$ and $\beta \in V^{*}$

if and only if

$\alpha = \alpha_1 \alpha_2 \alpha_3$, $\beta = \alpha_1 \beta_2 \alpha_3$ and $\alpha_2 \rightarrow \beta_2 \in P$

$\rightarrow$ α_2 is rewritten as β_2 in the context $<\alpha_1, \alpha_3>$

Example

In the grammar G

- $V_N = \{S, A, B, C, D\}$
- $V_T = \{a,b,c\}$
- S is the initial symbol
- $P = \{S \rightarrow AB, BA \rightarrow cCD, CBS \rightarrow ab, A \rightarrow \varepsilon\}$

- $aa\underline{\mathbf{BA}}S \Rightarrow aa\underline{\mathbf{cCD}}S$
- $bc\underline{\mathbf{CBS}}Add \Rightarrow bc\underline{\mathbf{ab}}Add$

Language generated by a grammar

- Given a grammar **G=$\langle V_N, V_T, P, S \rangle$,**
$L(G)=\{x \mid x \in V_T^* \wedge S \Rightarrow^+ x\}$
- Informally the language generated by a grammar G is **the set of all strings**
  - **Consisting only of terminal symbols**

  that can be **derived from S**
  - **In any number of steps**

Example 1

- G_1=<{S,A,B}, {a,b,0}, P, S>
  - $P=\{S \rightarrow aA,\ A \rightarrow aS,\ S \rightarrow bB,\ B \rightarrow bS,\ S \rightarrow 0\}$
- Some derivations
  - $S \Rightarrow 0$
  - $S \Rightarrow aA \Rightarrow aaS \Rightarrow aa0$
  - $S \Rightarrow bB \Rightarrow bbS \Rightarrow bb0$
  - $S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabb0$
- An easy generalization $L(G_1)=\{aa, bb\}^*.0$

Example 2

- G_2=<{S}, {a,b}, {$S \to aSb|ab$}, S>
  - {$S \to aSb|ab$} is an abbreviation for {$S \to aSb$, $S \to ab$}
- Some derivations
  - $S \Rightarrow ab$
  - $S \Rightarrow aSb \Rightarrow aabb$
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$
- An easy generalization $L(G_2)=\{a^n b^n | n>0\}$
  - $L(G_2)=\{a^n b^n | n \geq 0\}$ if we substitute $S \to ab$ with $S \to \varepsilon$

Example 3

- G_3=<{S,A,B, C, D}, {a,b,c}, P, S>
  - P={$S \rightarrow aACD$, $A \rightarrow aAC | \varepsilon$, $B \rightarrow b$, $CD \rightarrow BDc$, $CB \rightarrow BC$, $D \rightarrow \varepsilon$}
- Some derivations
  - $S \Rightarrow aACD \Rightarrow aCD \Rightarrow aBDc \Rightarrow^* abc$
  - $S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCBDc \Rightarrow aaBCDc \Rightarrow aabCDc \Rightarrow aabBDcc \Rightarrow aabbDcc \Rightarrow aabbcc$
  - $S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCCD \Rightarrow aaCC$

Some natural questions

- What is the **practical use** of grammars?
- **What languages** can be obtained through grammars?
- What is the **relationship between automata and grammars**?
  - And between languages generated by grammars and languages accepted by automata?
  - And the Chomsky hierarchy?