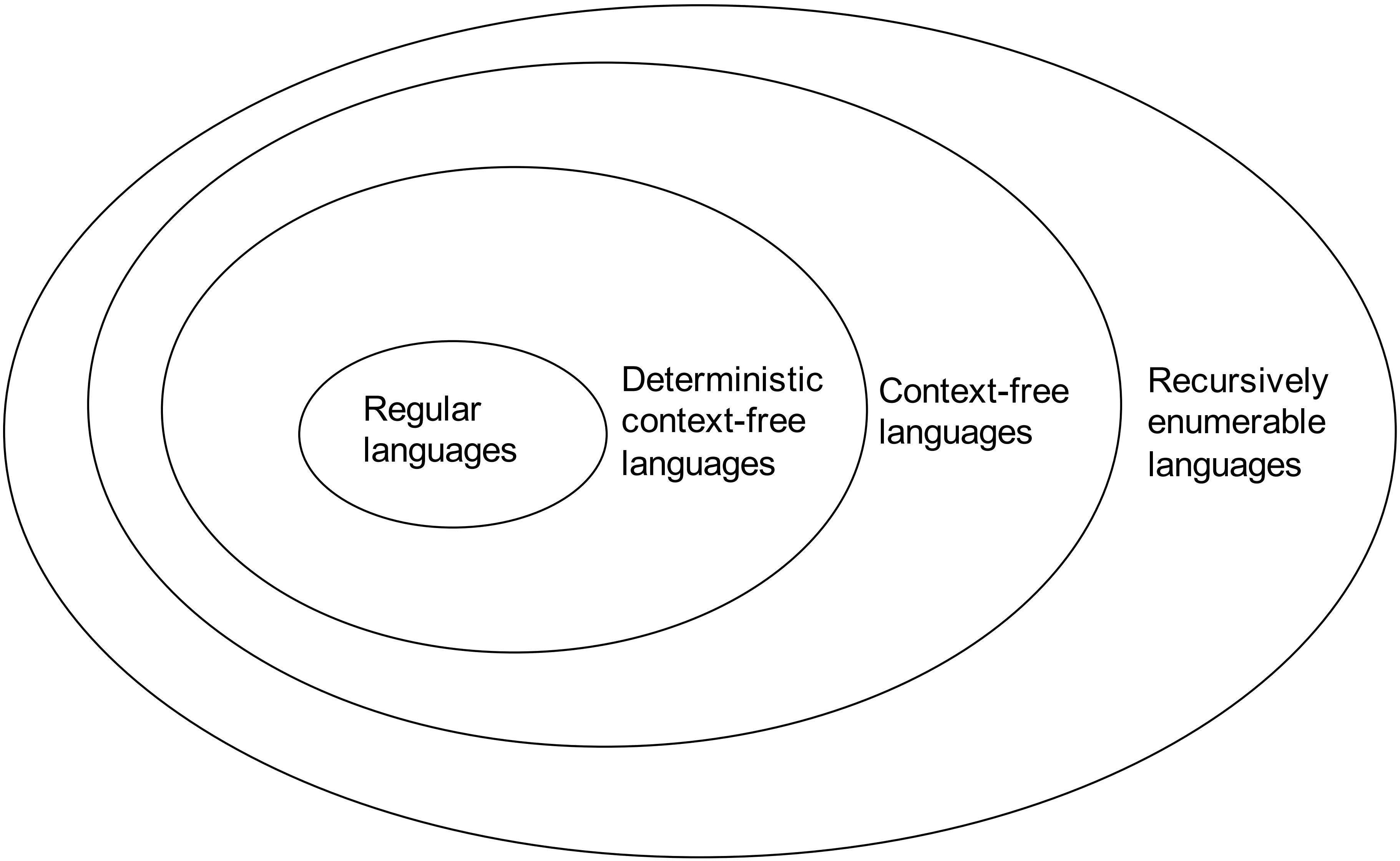
Theory of Computation

Generative Grammars

Lecture 11 - Manuel Mazzara

Languages - recap



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

Equivalences - recap

- **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 (lab session today)
- Thompson and Kleene algorithms plus several others **establish the equivalence of description formats for regular languages**

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 **"<u>main object</u>"**
  - Examples: a book, a program, a message, …

  as a sequence of "<u>composing objects</u>"
- Each "composing object" is "**<u>refined</u>**" by replacing it with more detailed objects and so on… until a sequence of **<u>base elements</u>** 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)

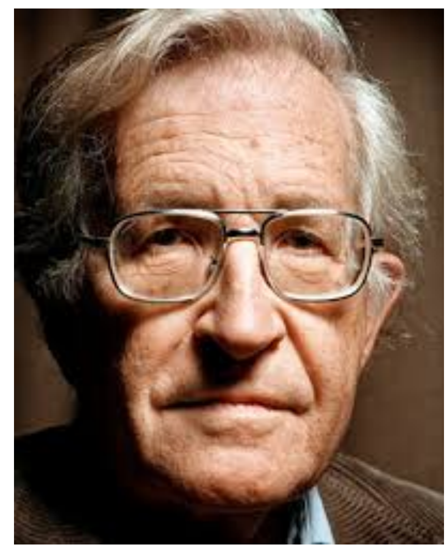
"A grammar can be regarded as **a device that enumerates the sentences of a language**"

"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

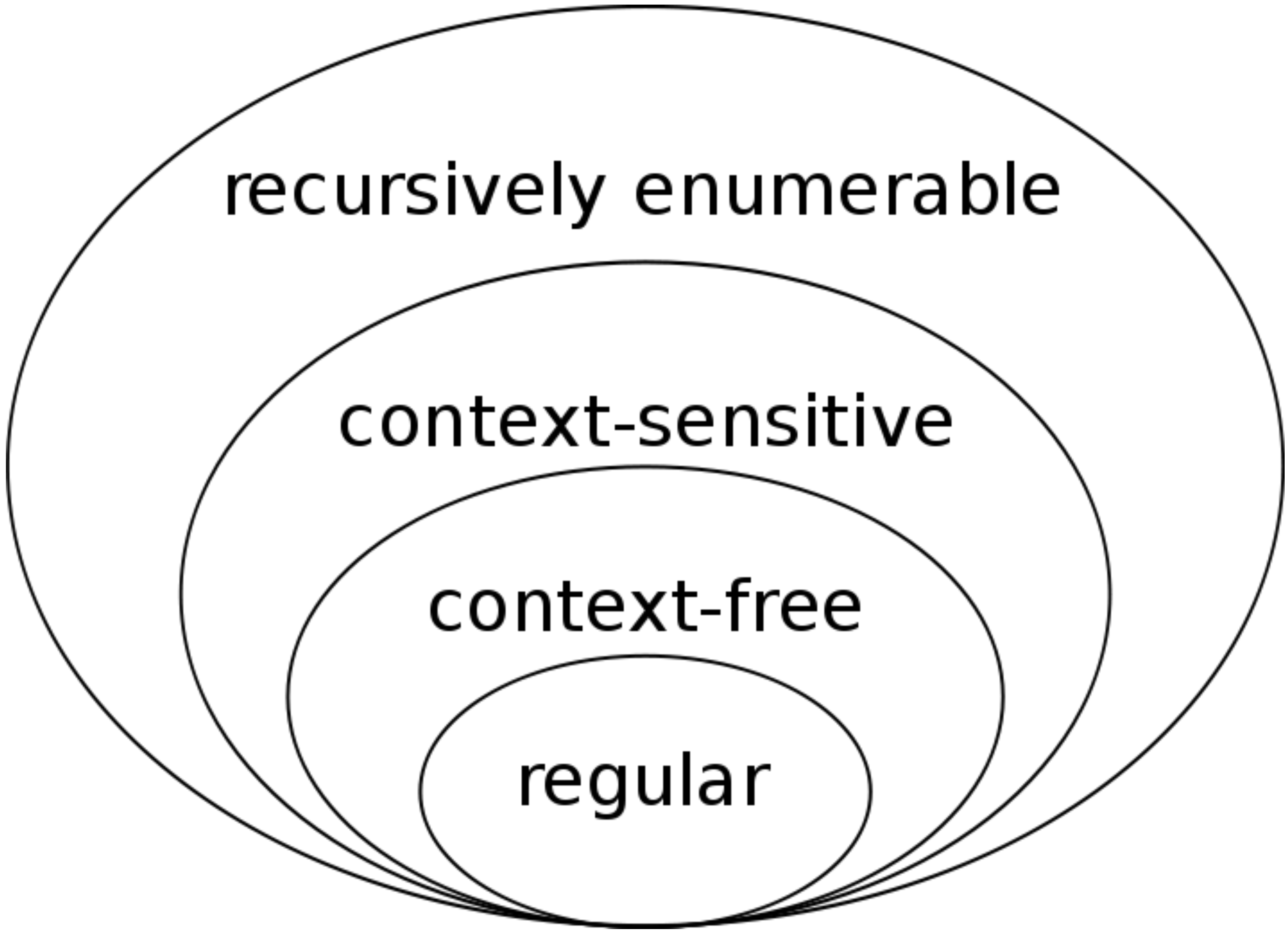


Noam Chomsky (2)

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

Productions

- A production is an element of $\mathbf{V^* \cdot V_N \cdot V^* \times V^*}$
  - This is usually denoted as $\boldsymbol{<\alpha, \beta>}$ where $\alpha \in \mathbf{V^* \cdot V_N \cdot V^*}$ and $\beta \in \mathbf{V^*}$
- We generally indicate a production as $\alpha \rightarrow \beta$
  - α **is a sequence of symbols including <u>at least one nonterminal</u> symbol** (it is a **<u>rewriting</u>** system)
  - β **is a (potentially empty) sequence of (terminal or non terminal) symbols**
  - We want to rewrite the left side into the right side, we need at least one nonterminal

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
- $P = \{S \rightarrow AB,$

  $BA \rightarrow cCD,$

  $CBS \rightarrow ab,$

  $A \rightarrow \varepsilon\}$

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

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 $\mathbf{G=<V_N, V_T, P, S>}$,
  $\mathbf{L(G)=\{x \mid x \in V_T^* \wedge S \Rightarrow^+ x\}}$
- 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

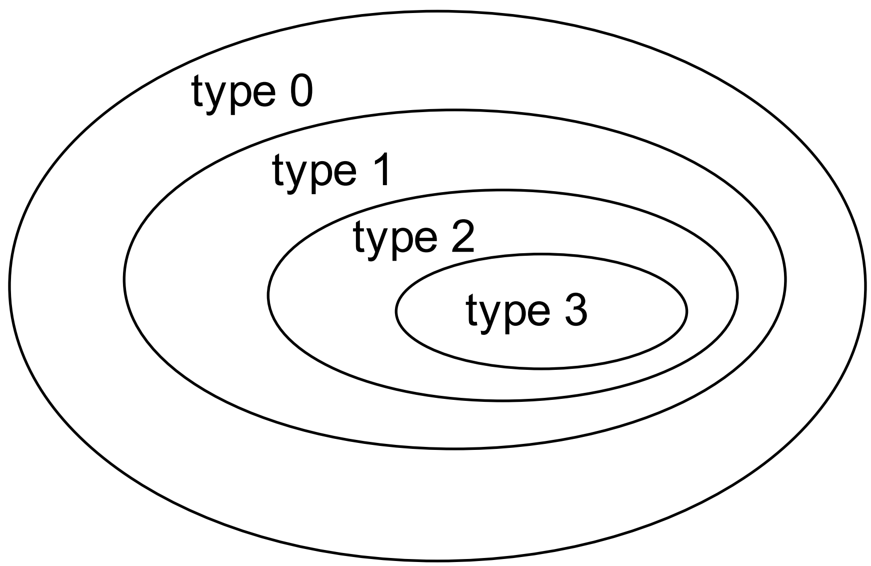
- G_2=<{S}, {a,b}, {$S \to aSb | ab$}, S>
  - {$S \to aSb | ab$} is an abbreviation for {$S \to aSb$, $S \to ab$}
- Some derivations
  - $S \Rightarrow ab$
  - $S \Rightarrow aSb \Rightarrow aabb$
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$
- An easy generalization $L(G_2)=\{a^n b^n | n>0\}$
  - $L(G_2)=\{a^n b^n | n\geq 0\}$ 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={$S \rightarrow aACD$, $A \rightarrow aAC \mid \varepsilon$, $B \rightarrow b$, $CD \rightarrow BDc$, $CB \rightarrow BC$, $D \rightarrow \varepsilon$}
- Some derivations
  - $S \Rightarrow aACD \Rightarrow aCD \Rightarrow aBDc \Rightarrow^* abc$
  - $S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCBDc \Rightarrow aaBCDc \Rightarrow aabCDc \Rightarrow aabBDcc \Rightarrow aabbDcc \Rightarrow aabbcc$
  - $S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCCD \Rightarrow aaCC$

Chomsky hierarchy

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



Unrestricted grammars (type 0)

Type-0 grammars include all formal grammars

$$\alpha \rightarrow \beta$$

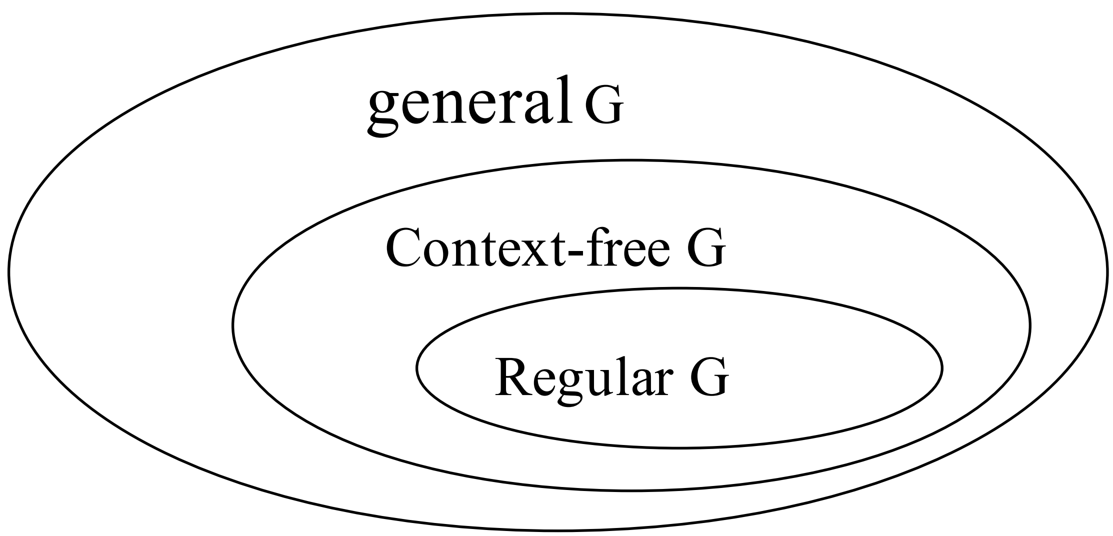
String of nonterminals and terminals

String of nonterminals and terminals

The only restriction on rules is : **left-hand side cannot be the empty string (you cannot generate symbols out of nothing)**

Definition

- General (also called unrestricted) grammars are grammars without any limitation on productions
  - They correspond to type 0 in the Chomsky hierarchy
- Both context-free grammars and regular grammars are non-restricted



Example (type 0)

VN = {S, T , C, P}

VT = {a, b}

P = {S → T E

T → aTa | bTb | C

C → CP

Paa → aPa

Pab → bPa

Pba → aPb

Pbb → bPb

PaE → Ea

PbE → Eb

CE → ε

Context-Sensitive grammars

- **Type-1 grammars** have rules of the form $\boldsymbol{\alpha A \beta \rightarrow \alpha\, \gamma\, \beta}$, 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
  - It should be clear why they are called context-sensitive…

Example (type 1)

- $V_N = \{S, A, B\}$
- $V_T = \{a,b,c\}$
- $P = \{S \rightarrow abc \mid aAbc ,$

  $Ab \rightarrow bA$

  $Ac \rightarrow Bbcc$

  $bB \rightarrow Bb$

  $aB \rightarrow aa$

  $aB \rightarrow aaA$

  $\}$

$L=\{a^n b^n c^n \mid n \geq 1\}$

Context-free grammars

- **Type-2 grammars** are defined by rules of the form $A \rightarrow \gamma$ where **A is a nonterminal** and γ **is a string of terminals and nonterminals**

Definition

- A grammar is called **context-free** (CFG) if
  - for each $\alpha \rightarrow \beta \in P$, we have $|\alpha| = 1$, i.e., α is an element of $V=V_N \cup V_T$
- They are called context-free because the rewriting of α does not depend on its context
  - context = part of the string surrounding it

Example (type 2)

- $V_N = \{S\}$
- $V_T = \{a,b\}$
- $P = \{S \rightarrow aSb \mid \varepsilon \}$

$L=\{a^n b^n \mid n \geq 0\}$

Context-free grammars

- **CFGs are the same as the BNFs used for defining the syntax of programming languages**
  - they are well fit to define typical features of programming and natural languages
  - Regular grammars are also context-free grammars
  - But not vice versa

Regular grammars

- **Type 3 grammars** 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: **right-linear XOR left-linear**) 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

Definition

- If for each $\alpha \to \beta \in P$ we have $|\alpha| = 1$ and
  - $\beta \in V_N . V_T \cup V_T$, the grammar is **left regular**
- If for each $\alpha \to \beta \in P$ we have $|\alpha| = 1$ and
  - $\beta \in V_T . V_N \cup V_T$, the grammar is **right regular**
- A grammar is **regular** (RG) iff it is left regular or right regular
- A language is regular iff it is generated by some regular grammar
  - There must be <u>at least ONE</u> grammar that generates it

Example (type 3)

- $V_N = \{S\}$
- $V_T = \{a\}$
- $P = \{S \rightarrow aS \mid \varepsilon \}$

$L=\{a^n \mid n \geq 0\}$

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?

Some answers

- Chomsky hierarchy can be "renamed"
  - Type 3 grammars: regular
  - Type 2 grammars: context-free
  - Type 1 grammars: context-sensitive
  - Type 0 grammars: unrestricted
- Correlations
  - Regular grammars – regular languages - FSAs
  - Context-free grammars – context-free languages -NDPDAs
  - Unrestricted grammars – recursively enumerable languages - MTs

Automata, languages, and grammars

Chomsky hierarchy	Grammars	Languages	Minimal automaton
Type-0	Unrestricted	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Context-sensitive	(Linear bounded automaton)
Type-2	Context-free	Context-free	NDPDA
Type-3	Regular	Regular	FSA

RGs and FSAs

Let A be a FSA. An equivalent RG G can be found constructively. Equivalent means that G generates exactly the same language that is recognized by A (and vice versa)

Regular grammars, finite state automata and regular expressions are different models to describe the same class of languages

Building a RG from a FSA

- If $A=\langle Q, I, \delta, q_0, F\rangle$, then it is possible to build $G=\langle V_N, V_T, S, P\rangle$ such that
  - $V_N = Q$,
  - $V_T = I$,
  - $S = \langle q_0\rangle$
  - For all $\delta(q, i) = q'$
    - $\langle q\rangle \rightarrow i\langle q'\rangle \in P$
    - If $q' \in F$ then $\langle q'\rangle \rightarrow \varepsilon \in P$
- $\delta^*(q, x) = q'$ if and only if $\langle q\rangle \Rightarrow^* x\langle q'\rangle$

Building a FSA from a RG

If $G=< V_N, V_T, S, P>$ then it is possible to build $A=<Q, I, \delta, q_0, F>$ such that

- $Q = V_N \cup \{q_F\}$
- $I = V_T$,
- $<q_0> = S$,
- $F = \{q_F\}$
- For all $A \rightarrow bC$, $\delta(A,b) = C$
- For all $A \rightarrow b$, $\delta(A,b) = q_F$

Automata, languages, and grammars

Chomsky hierarchy	Grammars	Languages	Minimal automaton
Type-0	Unrestricted	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Context-sensitive	(Linear bounded automaton)
Type-2	Context-free	Context-free	NDPDA
Type-3	Regular	Regular	FSA

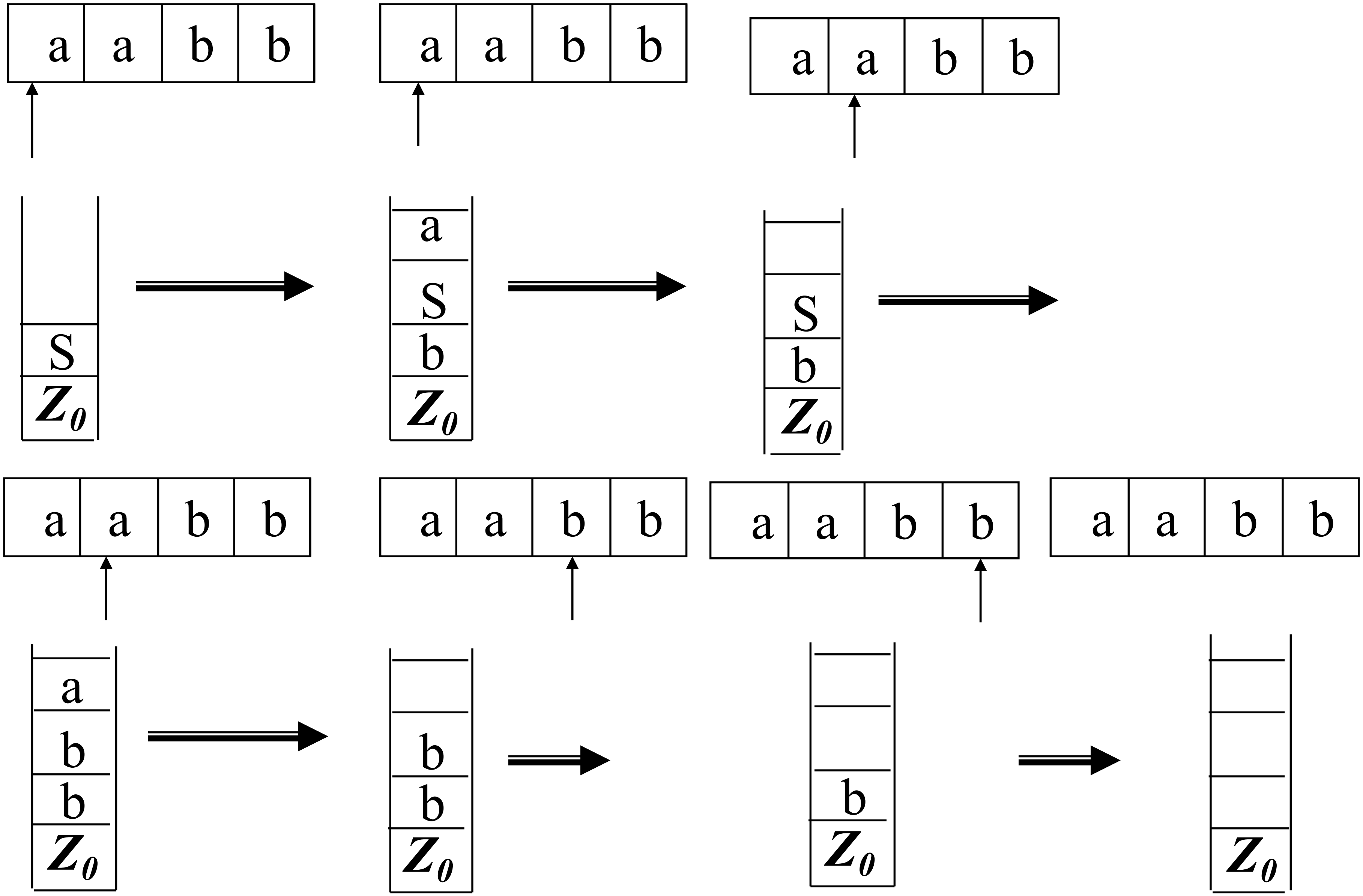
CFGs and NDPDAs

- Context-free grammars are equivalent to nondeterministic PDAs

- We show an intuitive justification

- The proof is the “core” of compiler construction

$$S \Rightarrow aSb \Rightarrow aabb$$



General grammars and TMs

- General grammars (GGs) and TMs are **equivalent formalisms** (constructive proof)
  - Given a GG it is possible to build a TM that recognizes the language generated by the grammar
  - Given a TM it is possible to define a GG that generates the language accepted by the TM
- How?

From a GG to a TM (1)

Given a general grammar **G=<V_N, V_T, P, S>,** let us construct a **NDTM** M such that **L(M)=L(G)** :

- **M has one memory tape**
- The **input string x is on the input tape**
- The **memory tape is initialized with S** (better: Z_0S)
- The **memory tape in general will contain a string** α ($\in V^*$)
  - It is scanned searching **the left part of some production of P**
  - When one is found, (not necessarily the first one) M operates a ND choice and **the chosen part is replaced by the corresponding right part** (if there are many right parts, again, M operates nondeterministically)

From a GG to a TM (2)

- In this way, whenever $\alpha \Rightarrow \beta$ we have $c_s = \langle q_s, Z_0\alpha \rangle \mid\text{-}^{*}\text{-} \langle q_s, Z_0\beta \rangle$
- **If and when the tape contains a string $y \in V_T^*$, it is compared with x**
  - If they coincide, **x is accepted**
  - otherwise this particular sequence of moves does not lead to acceptance

Remarks

- **Using a NDTM facilitates the construction but it is not necessary**
- Note that, if $x \notin L(G)$, M might "try an infinite number of ways"
  - some of these might never terminate, thus (correctly) being unable to conclude that $x \in L(G)$
  - *and* being unable to conclude $x \notin L(G)$

  Indeed the definition of acceptance requires that M reaches an accepting configuration if and only if $x \in L$
  - It does not require that M terminates its computation in a non-final state if $x \notin L$
  - Again, we have the complement problem and the asymmetry between solving a problem in the positive or negative sense

From a TM to a GG

- This is the opposite direction to show the full equivalence
- It is left as exercise
- It is a bit laborious, but conceptually simple