بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۲۱)

نظریهٔ زبان‌ها و ماشین‌ها

حسین فلسفین

Part 1: Linear Bounded Automata (LBA)

☞ *While it is not possible to extend the power of the standard Turing machine by complicating the tape structure, it is possible to limit it by restricting the way in which the tape can be used.*

☞ *A linear bounded automaton, like a standard Turing machine, has an unbounded tape, but how much of the tape can be used is a function of the input. In particular, we restrict the usable part of the tape to exactly the cells taken by the input.*

☞ *An LBA is a kind of Turing machine which is limited to operating within the space defined by the original placement of the input on the tape.*

☞ *We allow the machine to use only that part of the tape occupied by the input. Thus, more space is available for long input strings than for short ones, generating another class of machines, the linear bounded automata.*

An LBA is a restricted type of TM wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.

We can envision the input as bracketed by two special symbols, the left-end marker $[$ and the right-end marker $]$. For an input w, the initial configuration of the Turing machine is given by the instantaneous description $q_0[w]$. The end markers cannot be rewritten, and the read-write head cannot move to the left of $[$ or to the right of $]$.

نکات:

☞ *We omit the details of what a linear bounded automaton is, except to say that it is a somehow restricted nondeterministic single-tape Turing machine. The machine is never allowed to use cells outside of the region of cells on the tape where the input is first placed.*

☞ *Note that in this definition a linear bounded automaton is assumed to be nondeterministic. This is not just a matter of convenience but essential to the discussion of LBAs.*

☞ *Using a tape alphabet larger than the input alphabet allows the available memory to be increased up to a constant factor. Hence we say that for an input of length n, the amount of memory available is linear in n—thus the name of this model.*

☞ *Despite their memory constraint, linear bounded automata (LBAs) are quite powerful.*

Part 2: یک مسئلهٔ حل‌نشده (باز) معروف

Given language L accepted by some nondeterministic LBA, we don't know in general whether there is some deterministic LBA which also accepts L. This problem has remained open for many years despite the efforts of a good many talented and energetic people to solve it. In fact, it is an open question whether a nondeterministic LBA can recognize more languages than a deterministic LBA. (It is not known whether the deterministic LBA languages are a proper subset of the nondeterministic LBA languages.)

Part 3: Unrestricted Grammars

An unrestricted grammar (phrase-structure grammar) is a 4-tuple $G = (V, \Sigma, R, S)$, where V and Σ are disjoint sets of variables and terminals, respectively. S is an element of V called the start symbol, and R is a set of productions of the form $\alpha \to \beta$ where $\alpha, \beta \in (V \cup \Sigma)^*$ and α contains at least one variable.

☞ *The most general kind of grammars are the unrestricted grammars.*

☞ *In an unrestricted grammar, essentially no conditions are imposed on the productions. Any number of variables and terminals can be on the left or right, and these can occur in any order. There is only one restriction: α contains at least one variable.*

☞ *The class of languages they generate is exactly the class of languages that is accepted by TMs. Although we won't prove this statement. (Unrestricted grammars correspond to the largest family of languages so we can hope to recognize by mechanical means; that is, unrestricted grammars generate exactly the family of recursively enumerable languages.)*

☞ *Any language generated by an unrestricted grammar is recursively enumerable. (For every unrestricted grammar G, there is a Turing machine T with $L(T) = L(G)$.)*

☞ *For every recursively enumerable language L, there exists an unrestricted grammar G, such that $L = L(G)$. (For every Turing machine T with input alphabet Σ, there is an unrestricted grammar G generating the language $L(T) \subseteq \Sigma^*$.)*

☞ *We can continue to use much of the notation that was developed for CFGs. For example, $\alpha \Rightarrow_G^* \beta$ means that β can be derived from α in G, in zero or more steps, and $L(G) = \{x \in \Sigma^* | S \Rightarrow_G^* x\}$*

Part 4: Examples

Example: Let $L = \{a^{2^k} | k \in \mathbb{Z}^{\geq 0}\}$. L can be defined recursively by saying that $a \in L$ and that for every $n \geq 1$, if $a^n \in L$, then $a^{2n} \in L$. Using this idea to obtain a grammar means finding a way to double the number of a's in the string obtained so far. The idea is to use a variable D that will act as a "doubling operator." D replaces each a by two a's, by means of the production $Da \rightarrow aaD$. At the beginning of each pass, D is introduced at the left end of the string, and we think of each application of the production as allowing D to move past an a, doubling it in the process. The complete grammar has the productions

$$S \rightarrow LaR,\ L \rightarrow LD,\ Da \rightarrow aaD,$$
$$DR \rightarrow R,\ L \rightarrow \varepsilon,\ R \rightarrow \varepsilon.$$

Beginning with the string LaR, the number of a's will be doubled every time a copy of D is produced and moves through the string. Both variables L and R can disappear at any time. There is no danger of producing a string of a's in which the action of one of the doubling operators is cut short, because if R disappears when D is present, there is no way for D to be eliminated. The string $aaaa$ has the derivation

$$S \Rightarrow LaR \Rightarrow LDaR \Rightarrow LaaDR \Rightarrow LaaR \Rightarrow LDaaR$$
$$\Rightarrow LaaDaR \Rightarrow LaaaaDR \Rightarrow LaaaaR \Rightarrow aaaaR \Rightarrow aaaa$$

Example: Find a grammar G such that $L(G) = \{a^n b^n c^n | n > 0\}$.

Solution. The grammar G has nonterminals S, B, C, and the following rules:

$$S \longrightarrow aSBC \mid \varepsilon, \qquad CB \longrightarrow BC,$$
$$aB \longrightarrow ab, \qquad bB \longrightarrow bb,$$
$$bC \longrightarrow bc, \qquad cC \longrightarrow cc.$$

How does G generate $a^n b^n c^n$? It consists of three steps. First, it applies the first rule n times and the rule $S \to \varepsilon$ once to get a string $a^n (BC)^n$. Second, it applies the rule $CB \to BC$ for $n(n-1)/2$ times to move all B's to the left of all C's to get a string $a^n B^n C^n$. Finally, it uses the last four rules to change every B to b and every C to c.

$$\underline{S} \Rightarrow a\underline{S}BC \Rightarrow aa\underline{S}BCBC \Rightarrow aaa\underline{S}BCBCBC \Rightarrow aaaB\underline{CB}CBC$$
$$\Rightarrow aaaBBC\underline{CB}C \Rightarrow aaaBB\underline{CB}CC \Rightarrow aa\underline{aB}BBCCC$$
$$\Rightarrow aaa\underline{bB}BCCC \overset{*}{\Rightarrow} aaabbb\underline{bC}CC \Rightarrow aaabbbc\underline{CC}C \overset{*}{\Rightarrow} aaabbbccc.$$

Why don't we use the simpler rules $B \to b$ and $C \to c$ for the third step? That is because we do not want to let the grammar skip the second step and change all symbols B and C to b and c without first moving B's to the left of C's. Note that, using our grammar G, we are not able to change any B to b before moving it to the left of all C's. Thus, the grammar G does not generate any string not of the form $a^n b^n c^n$.

Example:

Find a grammar G such that $L(G) = \{ww \mid w \in \{a, b\}^*\}$.

Solution. The grammar G has $V = \{S, T, A, B, R, L_a, L_b, [\,,]\}$ and rules

$$
\begin{aligned}
S &\longrightarrow T\,], & T &\longrightarrow aTA \mid bTB \mid [\,R,\\
RA &\longrightarrow AR, & RB &\longrightarrow BR,\\
AR] &\longrightarrow L_a\,], & BR] &\longrightarrow L_b\,],\\
AL_a &\longrightarrow L_aA, & AL_b &\longrightarrow L_bA,\\
BL_a &\longrightarrow L_aB, & BL_b &\longrightarrow L_bB,\\
[\,L_a &\longrightarrow a[\,R, & [\,L_b &\longrightarrow b[\,R,\\
[\,R] &\longrightarrow \varepsilon.
\end{aligned}
$$

This grammar uses the same idea of moving a symbol R, L_a or L_b around to change the sentential forms. First, it uses the first two rules to get $S \overset{*}{\Rightarrow}$

$w\,[\,R\tilde{w}^R\,]$, where $\tilde{w}$ is the string w with symbol a replaced by A and symbol b replaced by B. Next, it uses symbol L_a or L_b to *carry* a terminal symbol a or b, respectively, to the left and put it to the left of the left-end mark [. This process reverses $\tilde{w}^R$ to w. The following is the derivation of $aabaaaba$:

$$\begin{aligned}
\underline{S} &\Rightarrow \underline{T}\,] \overset{*}{\Rightarrow} aaba\underline{T}ABAA\,] \Rightarrow aaba\,[\,\underline{RA}BAA\,] \overset{*}{\Rightarrow} aaba\,[\,ABA\underline{AR}\,] \\
&\Rightarrow aaba\,[\,AB\underline{AL_a}\,] \overset{*}{\Rightarrow} aaba\,[\,\underline{L_a}ABA\,] \Rightarrow aabaa\,[\,\underline{RA}BA\,] \\
&\overset{*}{\Rightarrow} aabaa\,[\,AB\underline{AR}\,] \Rightarrow aabaa\,[\,A\underline{BL_a}\,] \overset{*}{\Rightarrow} aabaa\,[\,\underline{L_a}AB\,] \\
&\Rightarrow aabaaa\,[\,\underline{RA}B\,] \overset{*}{\Rightarrow} aabaaaba\,[\,\underline{R}\,] \Rightarrow aabaaaba.
\end{aligned}$$

Again, it is easy to see that G cannot generate strings not of the form ww because each nonterminal has its fixed role and can only move around as designed. □

Part 5: Context-Sensitive Grammars (CSGs)

Definition: *A context-sensitive grammar (CSG) is an unrestricted grammar in which no production is length-decreasing. In other words, every production is of the form $\alpha \to \beta$, where $|\beta| \geq |\alpha|$.*

The definition allows no length-reducing rules at all. It is not possible to generate any language that contains ε. (A length-increasing grammar cannot generate the empty string.)

A language L is said to be context sensitive if there exists a context-sensitive grammar G, such that $L = L(G)$ or $L = L(G) \cup \{\varepsilon\}$. In other words, a language L is said to be context-sensitive (or a CSL) if $L - \{\varepsilon\}$ is generated by some CSG. (The funny condition involving arises because a CSG cannot have ε-productions, and hence, cannot generate ε.)

ارتباط بین LBAها و CSGها:

☞ *If L is accepted by a linear-bounded automaton, then there is a context-sensitive grammar G generating $L - \{\varepsilon\}$.*

☞ *If L is a context-sensitive language, then there is a linear-bounded automaton that accepts L.*

☞ *A language L is context-sensitive iff there exists an LBA that accepts it.*

جایگاه زبان‌های مستقل از متن در سلسله‌مراتب زبان‌ها:

Every CSL is recursive. There exists a recursive language that is not a CSL.

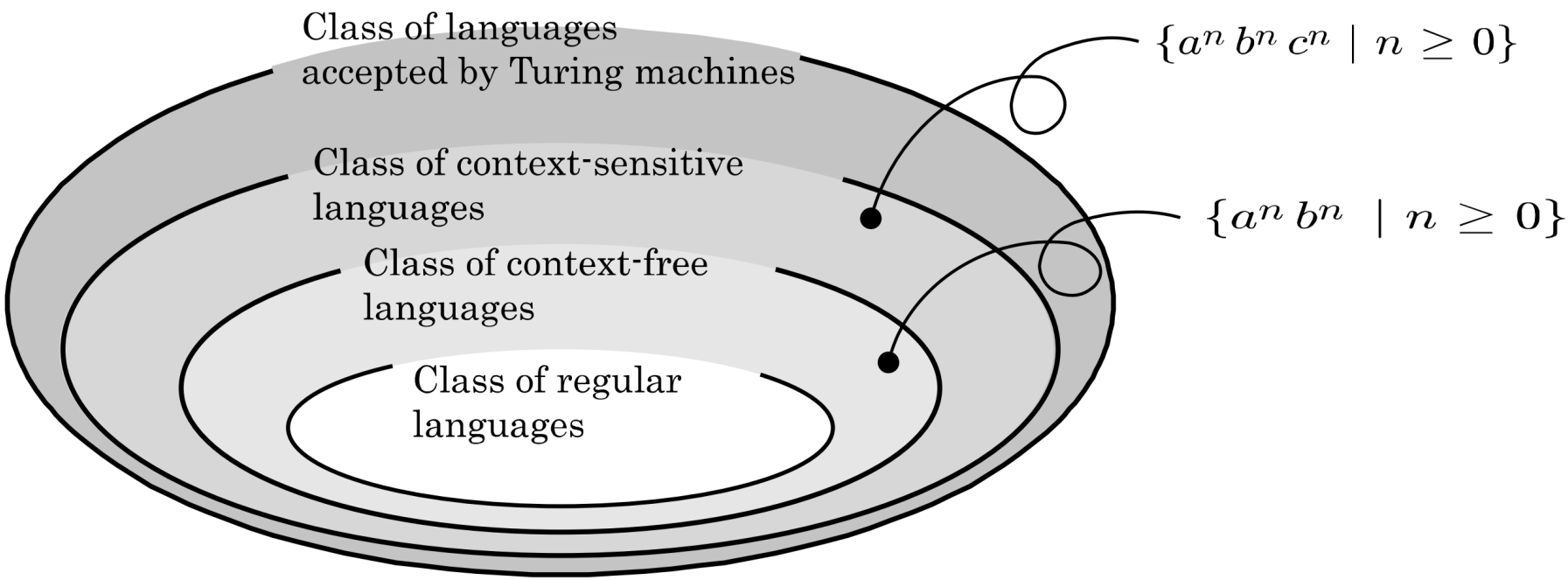
Part 6: Chomsky Hierarchy

Noam Chomsky (1928–),the influential American linguist and scholar, defined a hierarchy of language classes. It is now called the Chomsky hierarchy.

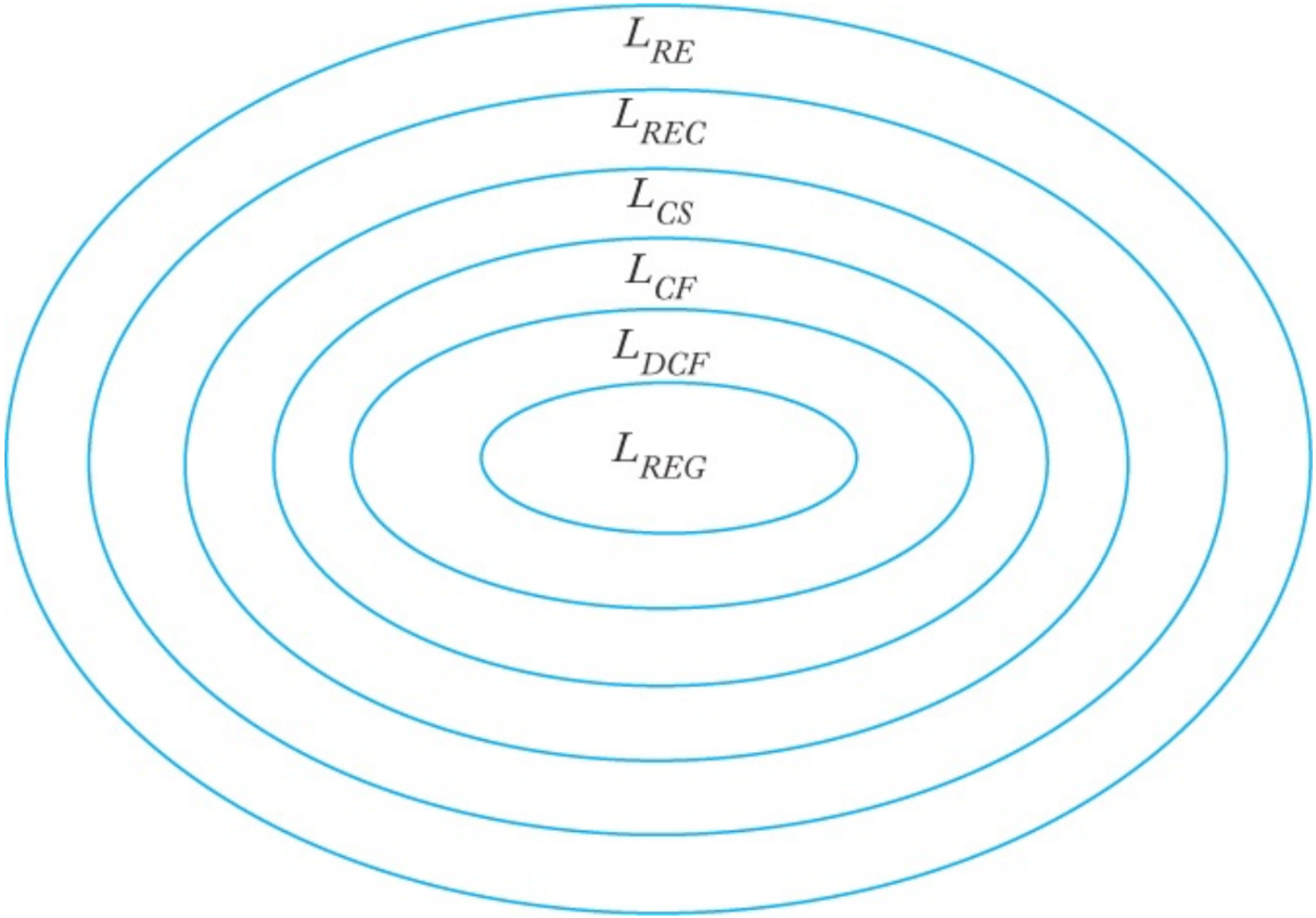
☞ *Note that nearly every class is represented by both a machine model and a grammar type. This "duality"is extremely useful in proving theorems, since one has the freedom to choose the appropriate representation. The exception is the class of recursive languages, which does not have a corresponding grammar model.*

☞ *Note that each class of languages is strictly contained in the one after it.*

The four classes of languages listed below constitute the Chomsky hierarchy:
finite automaton ⇔ regular grammar (type 3),
pushdown automaton ⇔ context-free grammar (type 2),
linear bounded automaton ⇔ context-sensitive grammar (type 1),
Turing machine ⇔ phrase structure grammar (type 0).



The extended hierarchy is shown in the following figure:



Type	Languages (Grammars)	Form of Productions in Grammar	Accepting Device
3	Regular	$A \to aB,\ A \to \Lambda$ $(A, B \in V,\ a \in \Sigma)$	Finite automaton
2	Context-free	$A \to \alpha$ $(A \in V,\ \alpha \in (V \cup \Sigma)^*)$	Pushdown automaton
1	Context-sensitive	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup \Sigma)^*,\ \lvert\beta\rvert \geq \lvert\alpha\rvert,$ α contains a variable)	Linear-bounded automaton
0	Recursively enumerable (unrestricted)	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup \Sigma)^*,$ α contains a variable)	Turing machine

Abbreviation	Language class	Machine model	Grammar
REG	Regular languages	DFA, NFA, NFA-ϵ, 2DFA	Regular grammar (aka Type 3)
DCFL	Deterministic context-free languages	Deterministic PDA	LR(k) grammar, $k \geq 1$
CFL	Context-free languages	PDA	Context-free grammar (aka Type 2)
CSL	Context-sensitive languages	LBA	Context-sensitive grammar, length-increasing grammar (aka Type 1)
REC	Recursive languages	Always-halting TM	
RE	Recursively enumerable languages	TM	Unrestricted grammar (aka Type 0)

تمرین: درایویشن نظیر رشتهٔ

$$aabbaaabba$$

را در گرامر **unrestricted** معرفی‌شده برای زبان

$$\{ww|w \in \{a,b\}^*\}$$

(اسلاید ۱۲) بنویسید.