4CS4-6: Theory of Computation (Chomsky Hierarchy Grammars and LBA)

Lecture 22: April, 09, 2020

Prof. K.R. Chowdhary *: Professor of CS*

22.1 Introduction

The Chomsky hierarchy, as originally defined by Noam Chomsky, comprises four types of languages and their associated grammars and machines. Table 22.1 shows the Chomsky hierarchy of grammars. These languages form a strict hierarchy, that is,

$$\begin{aligned}\textit{Regular languages} &\subseteq \textit{Context-free languages}\\ &\subseteq \textit{Context-sensitive languages}\\ &\subseteq \textit{Recursively enumerable languages.}\end{aligned}$$

Table 22.1: Chomsky Hierarchy

S. No.	Language	Grammar	Machine	Language Example
1.	Regular language (Type-3 language)	Regular grammar: Right-linear grammar/ Left-linear grammar	Deterministic or nondeterministic finite automata	a^*
2.	Context-free language (Type-2 language)	Context-free grammar	Nondeterministic-pushdown automata	$\{a^n b^n \mid n \geq 0\}$
3.	Context-sensitive language (Type-1 language)	Context-sensitive grammar	Linear-bounded automata	$\{a^n b^n c^n \mid n \geq 0\}$
4.	Recursively enumerable language (Type-0 language)	Unrestricted grammar	Turing machine	Any computable function

.

22.2 Context Sensitive Grammars and Languages

We have discussed other types of languages besides those in the "classical" Chomsky hierarchy. For example, we noted that nondeterministic pushdown automata (NPDA) were more

Table 22.2: Extended Chomsky Hierarchy

S.No.	Language	Machine
1.	Regular language	Deterministic or nondeterministic finite state acceptor
2.	Deterministic context-free language	Deterministic pushdown automaton
3.	Context-free language	Nondeterministic pushdown automaton
4.	Context-sensitive language	Linear-bounded automaton
5.	Recursive language	Turing machine that halts
6.	Recursively enumerable language	Turing machine

powerful than deterministic Pushdown Automata (DPDA). Table 22.2 shows some of the language classes that fit readily into this hierarchy.

Not all language classes fit neatly into a hierarchy. For example, the linear languages, which (like deterministic context-free languages) fit neatly between the regular languages and the context-free languages. However, there are languages that are linear but not deterministic context-free, and there are languages that are deterministic context-free but not linear.

In fact, mathematicians have defined dozens, maybe hundreds, of different classes of languages, and how these relate to one another. We should know at least the four "classic" categories that are taught in almost every textbook on the subject.

A family of grammars $G = (\Sigma, V, S, P)$ is defined by restrictions placed on the form of production rules. We have studied that a regular grammar is defined as consisting only productions of,

$$A \to aA \mid a \mid \varepsilon$$

where $A \in V$, $B \in V$, and $a \in \Sigma$.

A CFG is defined as,

$$A \to \alpha$$

where $A \in V$, and $\alpha \in (V \cup \Sigma)^*$.

Note the context along with the variable in the left-hand side of the productions. For example, in $Cb \to bC$, in left hand side, b is context for C.

Definition 22.1 Unrestricted Grammar. *An unrestricted grammar $G = (\Sigma, V, S, P)$ has productions of the form $u \to v$, where, $u \in (V \cup \Sigma)^*$ and $v \in (V \cup \Sigma)^*$, with at least one variable symbol in u.*

Example 22.2 *The language $\{a^n b^n c^n \mid n \geq 0\}$, which we know that it cannot be derived using context-free grammar, can be generated by following context-sensitive grammar.*

$$\begin{aligned} V &= \{S, A, C\} \\ \Sigma &= \{a, b, c\} \\ S &\to aAbc \mid \varepsilon \\ A &\to aAbC \mid \varepsilon \\ Cb &\to bC \\ Cc &\to cc. \end{aligned}$$

Applying the above rules, we get,

$$\begin{aligned} S &\Rightarrow aAbc \\ &\Rightarrow aaAbCbc \ ; \ \text{using } A \to aAbC \\ &\Rightarrow aabCbc \ ; \ \text{using } A \to \varepsilon \\ &\Rightarrow aabbCc \ ; \ \text{using } Cb \to bC \\ &\Rightarrow aabbcc. \ \ \text{using } Cc \to cc \end{aligned}$$

Definition 22.3 Context-sensitive grammar. *A* context-sensitive *(CS) grammar $G = (\Sigma, V, S, P)$ has productions of the form $u \to v$, where $u, v \in (V \cup \Sigma)^*$ and $|u| \leq |v|$, and u has got at least one variable symbol in it.*

Theorem 22.4 *Every context-free language is context-sensitive.*

Proof: The productions of a context-free grammar have the format $A \to v$, where $v \in (\Sigma \cup V)^*$. The productions of a context-sensitive grammar have the form $xAy \to xvy$. Thus, in $A \to v$, with x, y as ε, it is context-sensitive. ■

Theorem 22.5 *There exists a context-sensitive language that is not context-free.*

Proof: The language $\{a^n b^n c^n \mid n \geq 0\}$ is not context-free (we used a pumping lemma to show this). We can show that it is context-sensitive by providing an appropriate context-sensitive grammar. Following are the productions for one such grammar. Note that it is different set than in example 22.2, as ε-production has been removed here.

$$\begin{aligned} &S \to aXBC \mid aBC \\ &X \to aXBC \mid aBC \\ &CB \to BC, \quad aB \to ab \\ &bB \to bb, \quad bC \to bc, \quad cC \to cc \end{aligned}$$

Using these rules we can generate the string *aabbcc*.

$$
\begin{aligned}
S &\Rightarrow aXBC \text{ ;using } S \to aXBC \\
&\Rightarrow aaBCBC \text{ ;using } X \to aBC \\
&\Rightarrow aabCBC \text{ ;using } aB \to ab \\
&\Rightarrow aabBCC \text{ ;using } CB \to BC \\
&\Rightarrow aabbCC \text{ ;using } bB \to bb \\
&\Rightarrow aabbcC \text{ ;using } bC \to bc \\
&\Rightarrow aabbcc \text{ ;using } cC \to cc
\end{aligned}
$$

■

Are there different grammars for the same language $a^n b^n c^n$, as demonstrated in example 122.2, and theorem 22.5? Truly not. In such situations, where the language is same, a grammar can be transformed into another grammar, or both can be transformed into some standard format such that they are identical.

Theorem 22.6 *Every context-sensitive language is recursive.*

Proof: A context-sensitive grammar is noncontracting; moreover, for any integer n there are only a finite number of sentential forms of length n. Therefore, for any string w we can set a bound on the number of derivation steps required to generate w, hence a bound on the number of possible derivations. The string w is in the language if and only if one of these derivations produces w. ■

Theorem 22.7 *There exists a recursive language that is not context-sensitive.*

Proof: The proof is left as an exercise. ■

22.3 Linear bounded Automata

These were originally developed as models for actual computers rather than models for the computational process. They have become important in the theory of computation even though they have not emerged in applications to the extent which pushdown automata enjoy.

In 1960, Myhill introduced an automation model today known as deterministic linear bounded automaton. Shortly thereafter, Landweber proved that the languages accepted by a deterministic LBA are always context-sensitive. In 1964, Kuroda introduced the more general model of (nondeterministic) linear bounded automata, and showed that the languages accepted by them are precisely the context-sensitive languages.

A linear bounded automaton (LBA) is a multi-track Turing machine which has only one tape, and this tape is exactly the same length as the input. That seems quite reasonable. We allow the computing device to use just the storage it was given at the beginning of its computation[1]. As a safety feature, we shall employ end-markers ($<$ on the left and $>$ on the

[1]For example, space consumed by a C program is equal to size of program code in machine language, plus the sum of size of all the variables defined, provided that program does not create dynamic storage.

right) on our LBA tapes and never allow the machine to go past them. This will ensure that the storage bounds are maintained and help keep our machines from leaving their tapes.

Definition 22.8 Linear bounded Automata. *A Turing machine that uses only the tape space occupied by the input is called a linear-bounded automaton (LBA).*

At this point, the question of accepting sets arises. Let's have linear bounded automata accept just like Turing machines. Thus for LBA halting means accepting.

For these new machines computation is restricted to an area bounded by a constant (the number of tracks) times the length of the input. This is very much like a programming environment where the sizes of values for variables is bounded.

A linear bounded automaton (LBA) is a nondeterministic Turing machine $M = (Q, \Sigma, \Gamma, \delta, s, H)$ (see Fig. 22.1) such that:

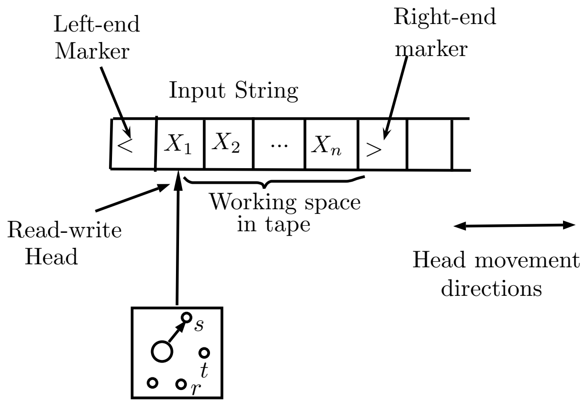

Figure 22.1: Physical model of Linear bounded Automata.

- There are two special tape symbols '$<$'and '$>$'(the left end marker and, right end marker).
- Σ is input alphabet, Γ is tape alphabet, $\Gamma = \Sigma \cup \{<, >\}$.
- The TM begins in the configuration $(s, < x >, 0)$, where s is start state, the R/W head points to 1st symbol of input x, and there is 0 or B after '$>$'end marker symbol.
- The TM cannot replace '$<$'or '$>$'with anything else, nor move the tape head left of '$<$'or right of '$>$'.
- $H = \{t, r\}$ is set of halting states, t is accept, and r is reject state.
- $\delta : (Q - \{t, r\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$.

When expressed as production, it is : $\alpha \to \beta$, with $|\alpha| \leq |\beta|$. And, the LBA is non-deterministic LBA.