Theory of Formal Languages (Linear bounded Automata)

Lecture 15: April 20, 2021

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

15.1 Introduction

The Linear Bounded Automata (LBA) 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 [myhill60]. Shortly thereafter, Landweber proved that the languages accepted by a deterministic LBA are always context-sensitive. In 1964, Kurodaintroduced the more general model of (nondeterministic) linear bounded automata, and showed that the languages accepted by them are precisely the context-sensitive languages.

15.2 Turing Machines and Grammars

Now we try to pay our attention to Turing machines as language accepters. We have already defined two classes of languages, i.e., recursive languages (R) and recursive enumerable languages (RE), depending on whether string membership in the respective languages can be decided on or merely accepted.

One approach to categorize the languages is using the (phrase structured) grammars. But, how do these language classes compare in relation to those defined by types of Turing Machines? We have seen as how to characterize these languages using the machines. For instance, recall that *regular languages* are recognized by finite state automata. We have also discussed that inverse of a regular language can be recognized by exchanging the final and non-final states of the corresponding finite automaton. In the following we shall show relation between Turing machine and regular languages, and that way between Turing machines and Finite automata.

15.2.1 Turing machines and Regular languages

We can simulate the FA (finite automata) using Turing machine, such that simulating Turing machine will only read the string w and tape-head will move only in right hand direction on tape. Such a Turing machine, after reaching to last symbol of input, will accept the input

if corresponding FA accepts this string else rejects. This shows that regular languages are accepted by TM. However, we have not been able to say any thing about halting of TM when it rejects the input w.

Consider the following approach. We construct Turing machine M that accepts the regular language $L = L(M)$, and for $w \in L$, it accepts and halts. Similarly, since there exists a FA that accepts language $\overline{L}$, the $\overline{L}$ is regular. We construct Turing machine $\overline{M}$ that accepts regular language $\overline{L} = L(\overline{M})$ and halts. As next step, we construct a Turing machine M' which combines Turing machines M and $\overline{M}$, such that $L' = L \cup \overline{L}$, and $L' = L(M')$. Because of construction of the Turing machine M', if M accepts input w and halts then M' accepts and halts, also if $\overline{M}$ accepts and halts then also M' accepts and halts. Thus, for both the situations, $w \in M$ and $w \notin M$ ($w \in \overline{M}$), the TM M' halts. This lead us to conclude that regular languages are in the class of *Recursive languages*, because they always make the Turing machine to halt. This observation is complete by noting that a Turing machine can easily act as a finite state automata, where all the transitions need to be of the form,

$$\delta(q_i, a) = (q_j, R)$$

where, we only read from the tape and transition internally from one state to the other, never writing on the tape or moving in any other directions. This brief and informal analysis allows us to affirm our intuition that regular languages (L_{Reg}) are included in Recursive languages (L_{Rec}), i.e., $\mathrm{L}_{Reg} \subseteq \mathrm{L}_{Rec}$.

15.2.2 Turing Machine and Context-Free Languages

The context-free languages are those that are generated by the context-free grammars, i.e., the grammars whose production rules are of the form $A \to (V \cup \Sigma)^*$, where A is variable/non-terminal symbol, V is set of variables, and Σ is set of terminal symbols. These languages are recognized by the machines, called Pushdown automata. Most of the machines we considered in are actually non-deterministic PDAs. We will make use of this information to show how the Turing machines can recognize CFLs.

The relation we would like to show is as follows: there is strict inclusion of CFL in recursive language. To show this we would have to demonstrate that:

1. There are recursive languages that are not context-free, and
2. Every CFL can be decided by some TM.

To show that first point in above is valid, we have to only show one case in favor of this point. The CFG can be converted to Chomsky Normal Form (CNF) where derivations of string w are bounded by $2|w| - 1$ steps. This is case of $\mathrm{L}_{\mathrm{CFG}} \subseteq \mathrm{L}_{\mathrm{Rec}}$. Other fact is, that $\{a^n b^n c^n \mid n \geq 0\}$ is in $\mathrm{L}_{\mathrm{Rec}}$ but, is not in $\mathrm{L}_{\mathrm{CFG}}$. This is case of context-free languages are subset of recursive languages, i.e., $\mathrm{L}_{\mathrm{CFG}} \subseteq \mathrm{L}_{\mathrm{Rec}}$. These shows that set of recursive languages is strictly larger than that of context-free languages. The language $\{a^n b^n c^n \mid n \geq 0\}$ satisfies our need of one case in favor. Note that we have shown that above language cannot be generated by any CFG, however we have shown that this language can be decided by a Turing machine .

This leaves us with the last task to show that CFL is decidable by a Turing machine. For this we try to prove a weaker result, namely that every CFL is included in the set of Recursively Enumerable Languages. We have following Lemma to prove it.

Lemma 15.1 *Every CFL can be recognized by some Turing Machine.*

Proof: We know that CFLs are recognized by Pudhdown automata. So we shall show that any PDA can be simulated by a Turing machine. We use 2-tape Turing machine, which works as follows:

- The first tape of the Turing machine simulated the tape of Pushdown automata, such that for the 1st tape of this Turing machine, the head moves only to right, without writing any thing,
- As the head moves on 1st tape, the head of 2nd tape may push a symbol on 2nd tape or it may pop it, based on the state the TM is in together with the symbol it has read from the first tape.

■

We have following detailed explanation of the Lemma above.

If a language is Context-free, then there exists an NPDA that can recognize it. Unfortunately, the non-determinism in NPDAs does not permit us to say much about the termination of every run of such a machine. What all NPDA recognition gives us is that there exists some sequence of transitions that accepts the strings in the recognized CFL. That is, we only have termination guarantee for strings in the language and the non-determinism of the machine prohibits us from stating any thing about decidability.

But, there is no problem, as we can use 2-tape Turing machine to easily simulate an NPDA, where 1st tape is used as the input tape to hold the input string undisturbed, and the head always moves to right in that. The 2nd tape of this Turing machine is used to simulate the NPDA stack, such that the symbols are added and removed from the right most position of the string on this 2nd tape. To keep the simulation simple, we can use a non-deterministic Turing machine. Such a simulation guarantees that there exists some deterministic Turing machine that can simulate the NPDA. Such a simulation guarantee that there exists some deterministic TM that can simulate the NPDA and therefore recognize the context-free language.

15.3 Linear-Bounded Automata

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 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.

[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 data defined, provided that program does not create dynamic storage.

Definition 15.2 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 (k_n, for $|w| = n$ and k tracks). 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. 15.1) such that:

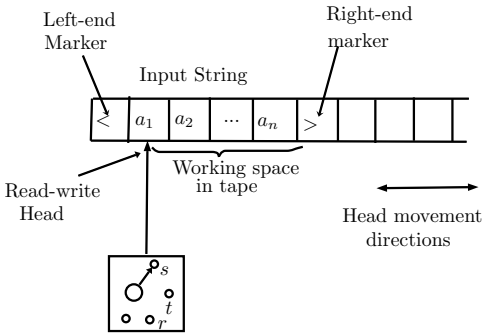

Figure 15.1: Physical model of Linear bounded Automata.

- 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 $x = a_1 a_2 ... a_n$, 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.
- $\delta : (Q - \{t, r\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$.

The productions of CSG (context sensitive grammar) are: $\alpha \to \beta$, with where $\alpha \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$, $|\alpha| \leq |\beta|$, and $\beta \in (V \cup \Sigma)^*$.

A linear-bounded automaton, as defined by Myhill (1960) [myhill60] and, following him, by Landweber [Landweber68], is a deterministic automaton specified by a tuple $(Q, \Sigma, \Gamma q_0, \delta, H)$, where the set Q of states, tape symbols Γ, and input alphabet Σ are finite sets, the initial state q_0 is an element of Q, the set H of final states is a subset of Q and, finally, the behavior function δ is a function from $(Q - H) \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$ where $\Gamma = \Sigma \cup \{B, <, >\}$,

satisfying the condition: if $\delta(<, s) = (<, s', k)$, where $k = R$. Here '<' is a symbol outside Γ and called the boundary symbol[2]. Similar is the case with right boundary symbol '>'.

The automaton works as follows. It is given, as input, a tape blocked into squares containing a string $< x >$, where x is a string in Σ and '<', '>'are the boundary symbols. Before operating on the input it is set in the initial state q_0. At the initial stage, then, it reads the left boundary in the state q_0. In general, if it is reading a symbol a in a state q, and if $\delta(a, q) = (a', q', k)$, it prints a' in the scanned square, and moves k squares to the right (i.e., one square to the right, one square to the left, or no move at all, according as $k = 1, -1$, or 0, respectively) and enters in the state q'. Continuing the calculation in this way, if it runs off the right end of the given tape and at this time it finds itself in one of the final states of H, then by definition the string x is *accepted*, or otherwise, *rejected*, by the automaton. The set of all strings accepted is the *language accepted* by the automaton.

Now, if we allow the behaviour function to be multivalued, we have a nondeterministic automaton. We understand, henceforth, by *linear-bounded automaton* a possibly nondeterministic automaton thus obtained. Only when there is possibility of confusion, we use the phrase "nondeterministic linear-bounded automaton" as a synonym of "linear-bounded automaton" . A linear-bounded automaton in Myhill's sense is referred to as a deterministic linear-bounded automaton. We then mean by a string accepted by a nondeterministic automaton M, a string for which there is a computation of M which, given the string as input, ends up off the right end of the tape in a final state. On the other hand, a string is said to be rejected by M if there is a computation of M which, given the string as input, never ends, or ends up off the left end of the tape, or, finally, ends up off the right end of the tape in a non-final state. Because of the nondeterminacy of M, a string can in general be both accepted and rejected by M . The set of all strings accepted by M is called the *languages accepted* by M. The set of all strings rejected by M is called the *language rejected* by M.

15.4 Language acceptability by LBA

In the followings, we prove some important proofs about LBA. Because of finite number of restricted length of the tape, the total configurations under which a LBA can go are finite, hence the languages accepted by LBA (i.e., Context sensitive languages) are decidable. On the same grounds it can be concluded that for LBA, the *Halting Problem* is decidable. Every LBA accepts context sensitive language (CSL), and for every CSL there exists an LBA that accepts it, i.e., a language is accepted by LBA if and only if it is CSL.

Theorem 15.3 *A language is accepted by an LBA if it is context sensitive.*

Proof: If L is a CSL, then L is accepted by some LBA.

1. Let $G = (V, \Sigma, S, P)$ be the given grammar such that $L(G) = L$, which is CSL, and we need to show that there is an LBA that recognizes L.
2. Construct LBA M with tape alphabet as pairs: $\Sigma \times \{V \cup \Sigma\}$ (for 2- track machine).

[2]This condition means that the symbol '<' is not effected during the computation

3. First track holds input string w, and second track holds a sentential form α of G, which started as start symbol S.
4. if $w = \epsilon$, M halts without accepting.
5. Repeat :
   (a) Non-deterministically select a position i in α.
   (b) Non-deterministically select a production $\beta \to \gamma$ of G.
   (c) If β appears beginning in position i of α, replace β by γ there.
   If the sentential form is longer than w, LBA halts without accepting.
   (d) Compare the resulting sentential form on track 2 with w on track 1. If they match, accept. If not go to step 1.

■

Example 15.4 *Show that the language $L = \{a^n b^n c^n \mid n \geq 0\}$ is context-sensitive.*

The production rules, we will be making in use are listed without any specific order as,

1. $S \to abc \mid aAbc$
2. $Ab \to bA$
3. $Ac \to Bbcc$
4. $bB \to Bb$
5. $aB \to aa \mid aaA$

For $w = aabbcc$, derivation can be obtained as follows:

$$S \Rightarrow aAbc \Rightarrow abAc \Rightarrow abBbcc$$
$$\Rightarrow aBbbcc \Rightarrow aabbcc$$

Note that above strings also be generated using the set of productions: $\{S \to SBc \mid abc, cB \to Bc, bB \to bb\}$. The reason for this being – we can always transform one grammar into other through simplification, normal forms, and reductions. We note that both of these grammars are context sensitive grammars.

The limitation (of size of storage) of LBA makes the LBA a somewhat more accurate model of computers that actually exists than a Turing machine, whose definition assumes unlimited tape.

Example 15.5 *Maximum number of configurations of an LBA.*

Let an LBA M has Q number of states, m number of symbols in the tape alphabets, and input of length n. Then, this M can have at most $\alpha(n)$ configurations, expressed by,

$$\alpha(n) = m^n * n * Q, \tag{15.1}$$

where m^n possible number of tape contents, n is possible number of head positions for given input of length n, and Q is number of states.

Theorem 15.6 *Every CSL is Recursive.*

Proof: Construct a 3-tape nondeterministic TM M to simulate the derivation of G.

1. First tape holds the input string.
2. Thirds tape is for the derivation.
3. Second tape holds the sentential form generated by the simulated derivation.

■

15.4.1 Space Bounds of LBA

The length restriction in CSGs has some intuition. What is the motivation for the restriction $|\alpha| \leq |\beta|$ in context-sensitive rules? The Idea is that, in a context-sensitive derivation $S \Rightarrow ... \Rightarrow ... \Rightarrow w$, all the sentential forms are of length at most $|w|$. This means that if L is context-sensitive, and we are trying to decide whether $w \in L$, we only need to consider possible sentential forms of length $\leq |w|$. So intuitively, we have the problem under control, at least in principle.

Note that, in the sentential form,

$$S = \alpha_a \Rightarrow \alpha_1 \Rightarrow ... \Rightarrow \alpha_i ... \Rightarrow \alpha_n = w,$$

there can be at the most only $(|\Gamma| + |V|)^{|w|}$ number of derivations, which is a finite number.

By contrast, without the length restriction, there is no upper limit on the length of intermediate forms that might appear in a derivation of w. So if we are searching for a derivation for w, how do we know when to stop looking? Intuitively, the problem here is wild and out of control. We will see this intuition made more precisely as we proceed.

The machine therefore has just a finite amount of memory, determined by the length of the input string. We call this a linear bounded automaton. An equivalent definition of an LBA is that it uses only k times the amount of space occupied by the input string, where k is a constant fixed for the particular machine. However, it is possible to simulate k tape cells with a single tape cell, by increasing the size of the tape alphabet.

At the bottom level of the Chomsky hierarchy, it makes no difference: every NFA can be simulated by a DFA. At the top level, the same happens. We have defined a "deterministic"

version of Turing machines – but any "nondeterministic Turing machine" can be simulated by a deterministic one.

At the context-free level, there is a difference – we need NPDA to account for all context-free languages. For example, $\Sigma^* - \{ww \mid w \in \Sigma^*\}$ is a context-free language whose complement is not context-free. However, if L is accepted by a DPDA then so is its complement – can just swap accepting and non-accepting states.

What about the context-sensitive level? Are NLBAs (nondeterministic LBAs) strictly more powerful than DLBAs? This is still an *open problem!* (Cannot use the same argument because it turns out that CSLs are closed under complementation – it was only shown in 1988.)

15.5 Turing Machine and Type-0 Grammars

In the following we shall show that a language is recognized by Turing machine if and only if it is generated by a type-0 grammar. To prove the "if" part, we construct a nondeterministic Turing machine that will non-deterministically choose a derivation in the grammar, and see whether or not the result of that derivation is the input string to the TM. If yes, the Turing machine accepts input. To show that the "only if" part, we construct a grammar which will non-deterministically generate the representation of a terminal string, and then simulate the TM on that string. If the string is accepted by the TM, the string is converted to the terminal symbols it represents.

We would like to show that a language is accepted by a TM iff it is generated by a type-0 grammar. This is proved through two theorems, as follows.

Theorem 15.7 *If a language L is generated by a type-0 grammar, then L is recognized by a Turing machine.*

Proof: Let $G = (V, \Sigma, S, P)$ be a type-0 grammar, and $L = L(G)$. We informally describe a Turing machine T accepting L, as follows. Let T be non-deterministic TM,

$$T = (Q, \Sigma, \Gamma, \delta, q_0, F), \tag{15.2}$$

where, $\Gamma = V \cup \Sigma \cup \{B, \#\}$.

In the above, the the symbols B, $\#$, S are not in $V \cup \Sigma$. We do not enumerate all the states in Q, but designate some of them when it is necessary.

To begin, T has input $w \in \Sigma^*$ on its tape. The T inserts $\#$ before w and shifts the entire w to right one cell. Then appends w with $\#S\#$, so that the tape contents becomes $\#w\#S\#$.

As next step, T will nondeterministically simulate a derivation in G starting with start symbol S. Each sentential form in the derivation will appear in turn between two '#' symbols, these symbols initially enclose S, and the new sentential produced replaces the S. If some choice of moves leads to a string of terminals there, that string is compared with w. If they match, T accepts w.

Formally, let T has $\#w\#A_1A_2...A_k\#$ on its tape, T moves its head through $A_1A_2...A_k$, non-deterministically choosing a position i and a constant r between 1 and the maximum length

of the left side of any production in P. Then T examines the substring $A_iA_{i+1}...A_{i+r-1}$. If $A_iA_{i+1}...A_{i+r-1}$ is the left-hand side of some production in P, it may be replaced by the right-hand side of that production. T may have to shift $A_{i+r}A_{i+r+1}...A_k\#$ either to left or right to make room to fill up space, should the right side of the production used have a length other than r.

From this simple simulation of derivation in G, it should be noted that T will print on its tape a string of the form $\#w\#\alpha\#$, and $\alpha \in (\Sigma \cup V)^*$ when $S \Rightarrow^* \alpha$. Also, if $\alpha = w$, then T accepts. ■

15.6 Chomsky Hierarchy Grammars and Languages

The Chomsky hierarchy, as originally defined by Noam Chomsky, comprises four types of languages and their associated grammars and machines. Table 15.6 shows the Chomsky hierarchy of grammars. These languages form a strict hierarchy, that is,

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

The table 15.6 characterize Chomsky-hierarchy of languages, and describes the nature of language and grammar for each machine. We note that just by varying the production rules, we can obtain all types of grammars and languages of Chomsky hierarchy. The following definition covers all four types of languages and grammars.

Definition 15.8 Grammars. *Type* $0, 1, 2, 3$.

For $i = 0, 1, 2, 3$, a grammar $G = (V, \Sigma, S, P)$ is of type i iff the restrictions (i) on productions as given below are satisfied:

For $i = 0, 1, 2, 3$, a language is of type i iff it is generated by a grammar of type i. The family of languages of type i is identified $\mathcal{L}_i$ [artsalo77].

In the followings, we describe each of these grammars and languages.

15.6.1 Chomsky Hierarchy

Type-3. These are called *Regular languages.* These are having grammars with production, where productions have following properties,

- there is one non-terminal in the LHS of each production,
- there is only one terminal in the right hand side,
- the RHS is having on-terminal and a terminal (left-linear) or terminal and a non-terminal (right-linear),

Table 15.1: 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-state automata	a^*
2.	Deterministic Context-free language (type-2 language)	Context-free grammar (There is no null symbol in productions, i.e., all productions are deterministic)	Deterministic Pushdown Automata	$\{a^n b^n \mid n \geq 1\}$
3.	Context-Free Language (type-2 language)	CFG (Each left-hand side has only one non-terminal)	Nondeterministic PDA	$\{ww \mid w \in \{a,b\}^*\}$,
4.	Context-Sensitive (type-1 language)	Context-Sensitive Grammar. For a production, $\|LHS\| \geq 1$, and $\|LHS\| \leq \|RHS\|$	Linear-bounded Automata	$\{a^n b^n c^n \mid n \geq 0\}$.
5.	Recursively Enumerable (tyoe-0) language.	Recursivley Enumerable Grammar. All the above grammars are allowed, and there can be $\|LHS\| \geq \|RHS\|$	Turing machine	Any computable function.

- the non-terminals are the states of a NFA that realizes the given regular languages.

These productions are expressed as,

$$X \to a, \text{ or}$$
$$X \to aY,$$

where $a \in \Sigma$, and $X, Y \in V$.

Type-2: Deterministic CFL. The corresponding DPDA and DCFLs parse the sentences without backtracking, i.e., deterministically. For example, in $\{a^n b^n \mid n \geq 0\}$, the language is deterministic, since we can deterministically switch to comparing b's. Each production in P is of the form,

$$X \to \alpha$$

where $X \to V$, and α is word over $V \cup \Sigma$.

Type-2: Context-Free Languages. These languages are more general than deterministic CFLs, since the restriction of determinism is removed in the corresponding machines and grammars. The language $L(G)$ for the grammar G: $S \to aSa \mid bSb \mid \varepsilon$, is non-deterministic CFL, because nondeterminism is required in order to guess the midpoint, for the corresponding language $L = \{ww^R \mid w \in \{a, b\}^*\}$ Also, the language $\{a^n b^m c^n d^m \mid n, m \geq 0\}$ is a CFL (i.e., nondeterministic).

Type-1: Context-Sensitive Languages. These are the languages which are recognized by linear bounded automata. The LBAs are restricted TMS, which can write only in the space equal to the length of input string w. In addition, the LBA can always decide whether the string w is accepted or not. However, the latter is not guaranteed for TM.

The context-sensitive grammar (CSG) have productions with left hand side with one or more non-terminals along with zero or more terminal symbols, but the length of the LHS of every production rule for that grammar is less or equal to the right hand side of that production. Such CSG specifies a pattern on the LHS and sentential form on the RHS. For example, we have rules in CSG like, $aBc \to abcd$ or $bBc \to Def$, which means in the first case when context is a, c the sentential for is $abcd$, in second case, when the context is b, c, the sentential is Def. These are the two different expansions of B, one when it is surrounded by a, c, and other when it is surrounded by b, c. Hence, the there is context sensitivity of the interpretation of B. The length of space used is restricted to $|w|$ to ensure the decidability of input sentence w.

Each production in P is of the form,

$$\alpha X \beta \to \alpha \gamma \beta$$

where, α, β are words over the alphabet $(V \cup \Sigma)$, $X \in V$, and γ is non-empty word over $(V \cup \Sigma)$. There is possible exception of production $X_0 \to \varepsilon$, whose occurrence in P implies, however, that X_0 does not occur on right side of any production in P.

Type-0: Recursive Enumerable (RE) languages. These languages are also called Turing recognition languages, and form the most general class of Chomsky Hierarchy. The TMS

recognize these languages, and the corresponding grammar have unrestricted form of productions. That is, they are like the productions in Context-Sensitive grammars, but there is no restriction like $|LHS| \leq |RHS|$ in the productions [gopal06].

The CFGs and CFLs are fundamentals to computer science because they describe the nested structure of all programming languages. The basic characteristic of CFLs is "nested structures", e.g., the nesting occurs in expressions (if...then...else) and in many other statically scoped structures in the computer programs.

We must distinguish the *iteration* from *nesting*. The basic characteristic of regular languages is "iteration" (i.e., looping). The C language program structure is and example of nesting.

$$\begin{array}{l} \textit{Lang-C} \rightarrow \textit{main} \quad () \quad \{\textit{braces}\} \\ \textit{braces} \rightarrow \{\textit{braces}\} \mid \varepsilon \end{array} .$$

15.6.2 Decidable Problems

We have introduced the Turing machine as a model of computation and studies it from the perspective of automata theory. The goal of the computation theory is to study the power and limitations of computations. In the following we discuss some of the problems that Turing machines can effectively solve, called *decidable languages*.

In the above we have discussed about various classes of languages: regular, context-free, recursive (decidable), and recursively enumerable languages, with following properties:

- every regular language is context-free,
- every regular language is decidable, and
- every decidable (recursive) language is recursively enumerable.

The Chomsky hierarchy contains five classes of formal languages, with a strict increasing subset relation between them. Each class of formal language is characterized by the class of machine deciding that class, or in equal sense the class of grammar generating it. The missing class of language is the set of context-sensitive languages. The Context-sensitive languages are accepted by linear bounded automata, which informally are Turing machines with finite tape length. It is thus easy to see that every context-sensitive languages are recursively enumerable. In fact every context-sensitive language is also decidable. We know that every context-free language is also context-sensitive. Thus, formally, we have the following relations:

- Every regular language is context-free,
- Every context-free language is context-sensitive,
- Every context-sensitive language is decidable,
- Every decidable language is recursively enumerable.

To show that every regular language is decidable, we simply use Turing machine to simulate a finite automaton. Consider that there is a regular language L and it is accepted by

a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$. Assume that M has exactly one accept state, $|F| = 1$. Given this, we construct deterministic Turing machine $M' = (Q', \Sigma', \delta', q_0', F')$ that can decide L. Also, $Q = Q'$, and $\Sigma = \Sigma'$. Also, for each $(\delta(q_i, a) = q_j) \in \delta$, there is $(\delta'(q_i, a) = q_j, a, R) \in \delta'$. Hence, Turing machine M' simulates DFA M, and $L(M') = L$. This notion can be rephrase in the form following theorem.

Theorem 15.9 *Let A_{DFA} be the language, whose every member is of the form $\langle M, w\rangle$, where M is DFA and w is string accepted by it, then $A_{DFA} = \{\langle M, w\rangle \mid M$ is DFA that accepts $w\}$ is decidable.*

Proof: We construct Turing machine M' to decide A_{DFA} as follows: $\langle M, w\rangle$ is input to Turing machine M', and it simulates DFA M on w, such that M' accepts $\langle M, w\rangle$ if and only if DFA M accepts w. Since every DFA is decider, hence M' is decider. Thus, Turing machine M' decides the A_{DFA}. ■