CSci 311, Models of Computation
Chapter 11
A Hierarchy of Formal Languages and Automata

H. Conrad Cunningham

29 December 2015

Contents

Acknowledgements: MS student Eli Allen assisted in preparation of these notes. These lecture notes are for use with Chapter 11 of the textbook: Peter Linz. *Introduction to Formal Languages and Automata*, Fifth Edition, Jones and Bartlett Learning, 2012.The terminology and notation used in these notes are similar to those used in the Linz textbook.This document uses several figures from the Linz textbook.

Advisory: The HTML version of this document requires use of a browser that supports the display of MathML. A good choice as of December 2015 seems to be a recent version of Firefox from Mozilla.

Introduction

The kinds of questions addressed in this chapter:

- What is the family of languages accepted by Turing machines?
- Are there any languages that are not accepted by any Turing machine?
- What is the relationship between Turing machines and various kinds of grammars?
- How can we classify the various families of languages and their relationships to one another?

Note: We assume the languages in this chapter are λ-free unless otherwise stated.

11.1 Recursive and Recursively Enumerable Languages

Here we make a distinction between languages accepted by Turing machines and languages for which there is a membership algorithm.

11.1.1 Aside: Countability

Definition (Countable and Countably Infinite): A set is *countable* if it has the same cardinality as a subset of the natural numbers. A set is *countably infinite* if it can be placed into one-to-one correspondence with the set of all natural numbers.

Thus there is some ordering on any countable set.

Also note that, for any finite set of symbols Σ, then Σ^* and any its subsets are countable. Similarly for Σ^+.

From Linz Section 10.4 (not covered in this course), we also have the following theorem about the set of Turing machines.

Linz Theorem 10.3 (Turing Machines are Countable): The set of all Turing machines is countably infinite.

11.1.2 Definition of Recursively Enumerable Language

Linz Definition 11.1 (Recursively Enumerable Language): A language L is *recursively enumerable* if there exists a Turing machine that accepts it.

This definition implies there is a Turing machine M such that for every $w \in L$

$$q_0 w \ \vdash_M^* \ x_1 q_f x_2$$

with the initial state q_0 and a final state q_f.

But what if $w \notin L$?

- M might halt in a nonfinal state.
- M might go into an infinite loop.

11.1.3 Definition of Recursive Language

Linz Definition 11.2 (Recursive Language): A language L on Σ is *recursive* if there exists a Turing machine M that accepts L and that halts on every w in Σ^*.

That is, a language is recursive if and only if there exists a *membership algorithm* for it.

11.1.4 Enumeration Procedure for Recursive Languages

If a language is recursive, then there exists an *enumeration procedure*, that is, a method for counting and ordering the strings in the language.

- Let M be a Turing machine that determines membership in a recursive language L on an alphabet Σ.
- Let M' be M modified to write the accepted strings to its tape.
- Σ^+ is countable, so there is some ordering of $w \in \Sigma^+$. Construct Turing machine $\hat{M}$ that generates all $w \in \Sigma^+$ in order, say $w_1, w_2, \cdots$.

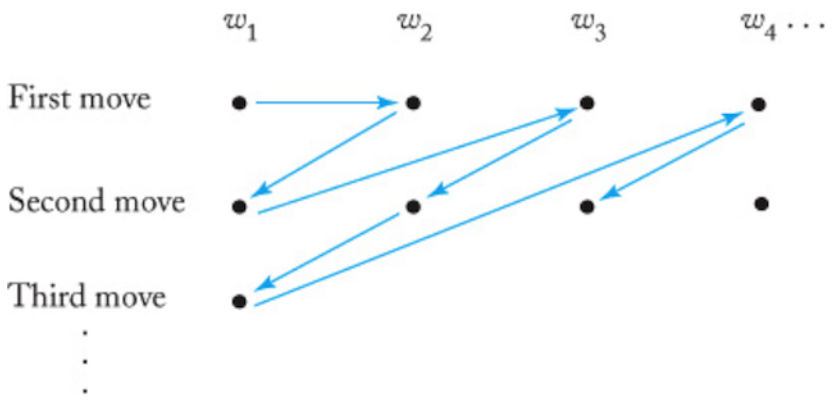
Thus $\hat{M}$ generates the candidate strings w_i in order. M' writes the the accepted strings to its tape in order.

11.1.5 Enumeration Procedure for Recursively Enumerable Languages

Problem: A Turing machine M might not halt on some strings.

Solution: Construct $\hat{M}$ to advance "all" strings simultaneously, one move at a time. The order of string generation and moves is illustrated in Linz Figure 11.1.

Now machine $\hat{M}$ advances each candidate string w_i (columns of Linz Figure 11.1) one M-move at a time.



Linz Fig. 11.1: Enumeration Procedure for Recursively Enumerable Languages

Because each string is generated by $\hat{M}$ and accepted by M in a finite number of steps, every string in L is eventually produced by M. The machine does not go into an infinite loop for a w_i that is not accepted.

Note: Turing machine $\hat{M}$ does not terminate and strings for which M does not halt will never complete processing, but any string that can be accepted by M will be accepted within a finite number of steps.

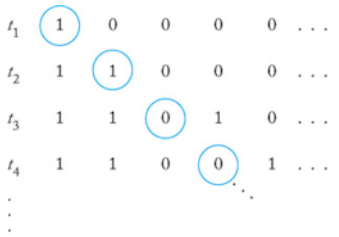
11.1.6 Languages That are Not Recursively Enumerable

Linz Theorem 11.1 (Powerset of Countable Set not Countable) Let S be an countably infinite set. Then its powerset 2^S is not countable.

Proof: Let $S = \{ s_1, s_2, s_3, \cdots \}$ be an countably infinite set.

Let $t \in 2^S$. Then t can represented by a bit vector $b_1 b_2 \cdots$ such that $b_i = 1$ if and only if $s_i \in t$.

Assume 2^S is countable. Thus 2^S can be written in order $t_1, t_2, \cdots$ and put into a table as shown in Linz Figure 11.2.



Linz Fig. 11.2: Cantor's Diagonalization

Consider the main diagonal of the table (circled in Linz Figure 11.2). Complement the bits along this diagonal and let t_d be a set represented by this bit vector.

Clearly $t_d \in 2^S$. But $t_d \neq t_i$ for any i, because they differ at least at s_i. This is a contradicts the assumption that 2^S is countable.

So the assumption is false. Therefore, 2^S is *not* countable. QED.

This is *Cantor's diagonalization* argument.

Linz Theorem 11.2 (Existence of Languages Not Recursively Enumerable): For any nonempty Σ, there exist languages that are not recursively enumerable.

Proof: Any $L \subseteq \Sigma^*$ is a language on Σ. Thus 2^{Σ^*} is the set of all languages on Σ.

Because Σ^* is infinite and countable, Linz Theorem 11.1 implies that the set of all languages on Σ is *not* countable. From Linz Theorem 10.3 (see above), we know the set of Turing machines can be enumerated. Hence, the recursively enumerable languages are countable.

Therefore, some languages on Σ are *not* recursively enumerable. QED.

11.1.7 A Language That is Not Recursively Enumerable

Linz Theorem 11.3: There exists a recursively enumerable language whose complement is not recursively enumerable.

Proof: Let $\Sigma = \{a\}$.

Consider the set of all Turing machines with input alphabet Σ, i.e., $\{M_1, M_2, M_3, \cdots\}$.

By Linz Theorem 10.3 (see above), we know that this set of is countable. So it has some order.

For each M_i there exists a recursively enumerable language $L(M_i)$.

Also, for each recursively enumerable languages on Σ, there is some Turing machine that accepts it.

Let $L = \{a^i : a^i \in L(M_i)\}$.

L is recursively enumerable because here is a Turing machine that accepts it. E.g., the Turing machine works as follows:

- Count a's in the input w to get i.
- Use Turing machine M_i to accept w.
- The combined Turing machine thus accepts L.

Now consider $\bar{L} = \{a^i : a^i \notin L(M_i)\}$.

Assume $\bar{L}$ is recursively enumerable.

There must be some Turing machine M_k, for some k, that accepts $\bar{L}$. Hence, $\bar{L} = L(M_k)$.

Consider a^k. Is it in L? Or in $\bar{L}$?

Consider the case $a^k \in \bar{L}$. Thus $a^k \in L(M_k)$. Hence, $a^k \in L$ by the definition of L. This is a contradiction.

Consider the case $a^k \in L$, i.e., $a^k \notin \bar{L}$. Thus $a^k \notin L(M_k)$ by definition of $\bar{L}$. But from the defintion of L, $a^k \in \bar{L}$. This is also be a contradiction.

In all cases, we have a contradiction, so the assumption is false. Therefore, $\bar{L}$ is not recursively enumerable. QED.

11.1.8 A Language That is Recursively Enumerable but Not Recursive

Linz Theorem 11.4: If a language L and its complement $\bar{L}$ are both recursively enumerable, then both languages are recursive. If L is recursive, then $\bar{L}$ is also recursive, and consequently both are recursively enumerable.

Proof: See Linz Section 11.2 for the details.

Linz Theorem 11.5: There exists a recursively enumerable language that is not recursive; that is, the family of recursive languages is a proper subset of the family of recursively enumerable languages.

Proof: Consider the language L of Linz Theorem 11.3.

This language is recursively enumerable, but its complement is not. Therefore, by Linz Theorem 11.4, it is not recursive, giving us the required example. QED.

There are well-defined languages that have no membership algorithms.

11.2 Unrestricted Grammars

Linz Definition 11.3 (Unrestricted Grammar): A grammar $G = (V, T, S, P)$ is an *unrestricted gramar* if all the productions are of the form

$$u \to v,$$

where u is in $(V \cup T)^+$ and v is in $(V \cup T)^*$.

Note: There is no λ on left, but otherwise the use of symbols is unrestricted.

Linz Theorem 11.6 (Recursively Enumerable Language for Unrestricted Grammar): Any language generated by an unrestricted grammar is recursively enumerable.

Proof: See Linz Section 11.2 for the details.

The grammar defines an enumeration procedure for all strings.

Linz Theorem 11.7 (Unrestricted Grammars for Recursively Enumerable Language): For every recursively enumerable language L, there exists an unrestricted grammar G, such that $L = L(G)$.

Proof: See Linz Section 11.2 for the details.

11.3 Context-Sensitive Grammars and Languages

Between the restricted context-free grammars and the unrestricted grammars, there are a number of kinds of "somewhat restricted" families of grammars.

Linz Definition 11.4 (Context-Sensitive Grammar): A grammar $G = (V, T, S, P)$ is said to be *context-sensitive* if all productions are of the form

$$x \to y,$$

where $x, y \in (V \cup T)^+$ and

$$|x| \leq |y|.$$

This type of grammar is *noncontracting* in that the length of successive sentential forms can never decrease.

All such grammars can be rewritten in a normal form in which all productions are of the form

$$xAy \to xvy.$$

This is equivalent to saying that the production

$$A \to v$$

can only be applied in a *context* where A occurs with string x on the left and string y on the right.

Linz Definition 11.5 (Context-Sensitive) : A language L is said to be *context-sensitive* if there exists a context-sensitive grammar G, such that $L = L(G)$ or $L = L(G) \cup \{\lambda\}$.

Note the special cases for λ. This enables us to say that the family of context-free languages is a subset of the family of context-sensitive languages.

11.3.1 Linz Example 11.2

The language $L = \{a^n b^n c^n : n \geq 1\}$ is a context-sensitive language. We show this by defining a context-sensitive grammar for the language, such as the following:

$S \to abc \mid aAbc$
$Ab \to bA$
$Ac \to Bbcc$
$bB \to Bb$
$aB \to aa \mid aaA$

Consider a derivation of $a^3b^3c^3$:

S	$\Rightarrow aAbc \Rightarrow abAc \Rightarrow abBbcc$
	$\Rightarrow aBbbcc \Rightarrow aaAbbcc \Rightarrow aabAbcc$
	$\Rightarrow aabbAcc \Rightarrow aabbBbccc \Rightarrow aabBbbccc$
	$\Rightarrow aaabbbccc$

The grammar uses the variables A and B as messengers.

- An A is created on the left, travels to the right to the first c, where it creates another b and c.
- Messanger B is sent back to the left to create the corresponding a.

The process is similar to how a Turing machine would work to accept the language L.

L is not context-free.

11.3.2 Linear Bounded Automata (lba)

In Linz Section 10.5 (not covered in this course), a *linear-bounded automaton* is defined as a nondeterministic Turing machine that is restricted to the part of its tape occupied by its input (bounded on the left by [and right by]).

[_____].

Linz Theorem 11.8: For every context-sensitive language L not including λ, there exists some linear bounded automaton M such that $L = L(M)$:

Proof: See Linz Section 11.3 for the details.

Linz Theorem 11.9: If a language L is accepted by some linear bounded automaton M, then there exists a context-sensitive grammar that generates L.

Proof: See Linz Section 11.3 for the details.

11.3.3 Relation Between Recursive and Context-Sensitive Languages

Linz Theorem 11.10: Every context-sensitive language L is recursive.

Linz Theorem 11.11: There exists a recursive language that is not context-sensitive.

We have studied a number of automata in this course. Ordered by decreasing power these include:

- Turing machine (accept recursively enumerable languages)
- linear-bounded automata (accept context-sensitive languages)

- npda (accept context-free languages)
- dpda (accept deterministic context-free languages)
- nfa, dfa (accept regular languages)

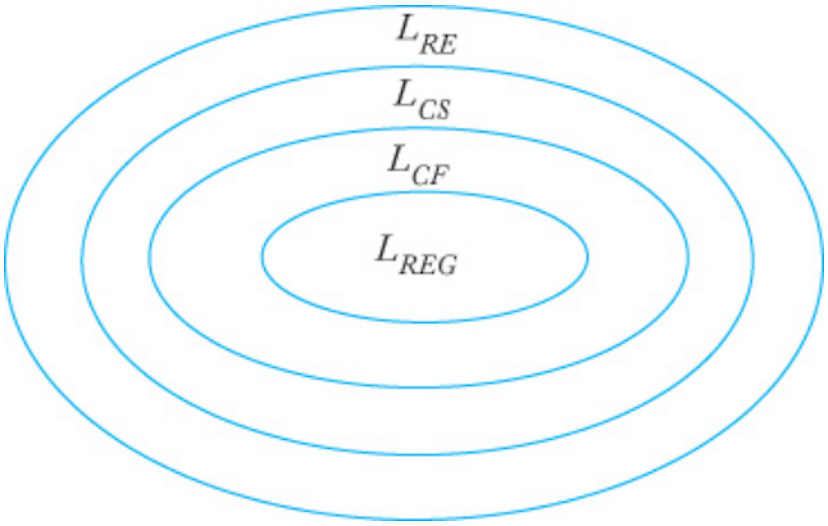
11.4 The Chomsky Hierarchy

We have studied a number of types of languages in this course, including

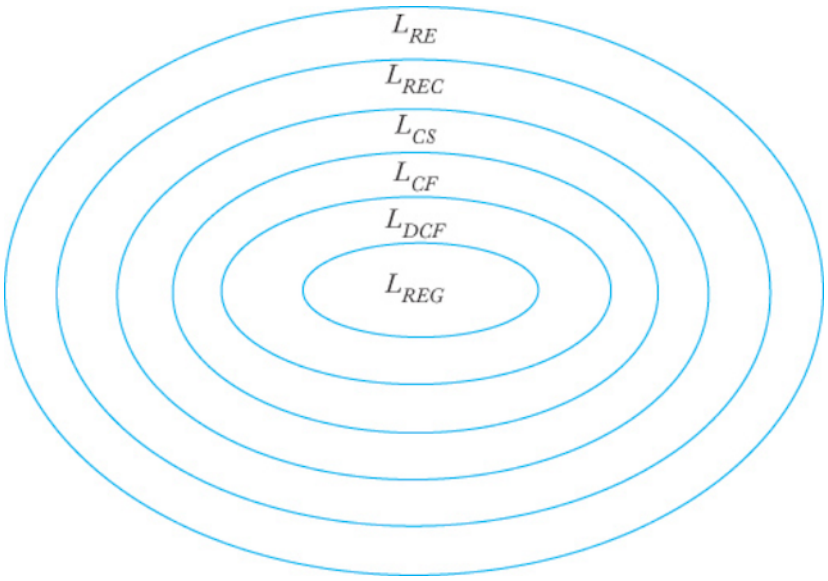
0. recursively enumerable languages L_{RE}
1. context-sensitive languages L_{CS}
2. context-free languages L_{REG}
3. regular languages L_{REG}

One way of showing the relationship among these families of languages is to use the *Chomsky hierarchy*, where the types are numbered as above and as diagrams in Linz Figures 11.3 and 11.4.

This classification was first described in 1956 by American linguist Noam Chomsky, a founder of formal language theory.



Linz Fig 11.3: Original Chomsky Hierarchy



Linz Fig 11.4: Extended Chomsky Hierarchy