CHAPTER 7

Regular Grammars ●

So far, we have considered two equivalent ways to describe exactly the class of regular languages:

- Finite state machines.
- Regular expressions.

We now introduce a third:

- Regular grammars (sometimes also called right linear grammars).

7.1 Definition of a Regular Grammar

A ***regular grammar*** G is a quadruple (V, Σ, R, S), where:

- V is the rule alphabet, which contains nonterminals (symbols that are used in the grammar but that do not appear in strings in the language) and terminals (symbols that can appear in strings generated by G),
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite set of rules of the form $X \rightarrow Y$, and
- S (the start symbol) is a nonterminal.

In a regular grammar, all rules in R must:

- have a left-hand side that is a single nonterminal, and
- have a right-hand side that is ε or a single terminal or a single terminal followed by a single nonterminal.

So $S \rightarrow \mathrm{a}$, $S \rightarrow \varepsilon$, and $T \rightarrow \mathrm{a}S$ are legal rules in a regular grammar. $S \rightarrow \mathrm{a}S\mathrm{a}$ and $\mathrm{a}S\mathrm{a} \rightarrow T$ are not legal rules in a regular grammar.

We will formalize the notion of a grammar generating a language in Chapter 11, when we introduce a more powerful grammatical framework, the context-free grammar. For now, an informal notion will do. The language generated by a grammar $G = (V, \Sigma, R, S)$, denoted $L(G)$, is the set of all strings w in Σ^* such that it is possible to start with S, apply some finite set of rules in R, and derive w.

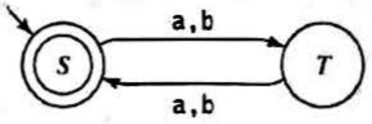
To make writing grammars easy, we will adopt the convention that, unless otherwise specified, the start symbol of any grammar G will be the symbol on the left-hand side of the first rule in R_G.

EXAMPLE 7.1 Even Length Strings

Let $L = \{w \in \{\mathrm{a}, \mathrm{b}\}^*: |w| \text{ is even}\}$. The following regular expression defines L:

$$((\mathrm{aa}) \cup (\mathrm{ab}) \cup (\mathrm{ba}) \cup (\mathrm{bb}))^*.$$

The following DFSM M accepts L:



The following regular grammar G also defines L:

$$S \rightarrow \varepsilon$$
$$S \rightarrow \mathrm{a}T$$
$$S \rightarrow \mathrm{b}T$$
$$T \rightarrow \mathrm{a}S$$
$$T \rightarrow \mathrm{b}S$$

In G, the job of the nonterminal S is to generate an even length string. It does this either by generating the empty string or by generating a single character and then creating T. The job of T is to generate an odd length string. It does this by generating a single character and then creating S. S generates ε, the shortest possible even length string. So, if T can be shown to generate all and only the odd length strings, we can show that S generates all and only the remaining even length strings. T generates every string whose length is one greater than the length of some string S generates. So, if S generates all and only the even length strings, then T generates all and only the other odd length strings.

Notice the clear correspondence between M and G, which we have highlighted by naming M's states S and T. Even length strings drive M to state S. Even length strings are generated by G starting with S. Odd length strings drive M to state T. Odd length strings are generated by G starting with T.

7.2 Regular Grammars and Regular Languages

THEOREM 7.1 Regular Grammars Define Exactly the Regular Languages

Theorem: The class of languages that can be defined with regular grammars is exactly the regular languages.

Proof: We first show that any language that can be defined with a regular grammar can be accepted by some FSM and so is regular. Then we must show that every regular language (i.e., every language that can be accepted by some FSM) can be defined with a regular grammar. Both proofs are by construction.

Regular grammar* → *FSM: The following algorithm constructs an FSM M from a regular grammar $G = (V, \Sigma, R, S)$ and assures that $L(M) = L(G)$:

grammartofsm(G: regular grammar) =

1. Create in M a separate state for each nonterminal in V.
2. Make the state corresponding to S the start state.
3. If there are any rules in R of the form $X \rightarrow w$, for some $w \in \Sigma$, then create an additional state labeled #.
4. For each rule of the form $X \rightarrow wY$, add a transition from X to Y labeled w.
5. For each rule of the form $X \rightarrow w$, add a transition from X to # labeled w.
6. For each rule of the form $X \rightarrow \varepsilon$, mark state X as accepting.
7. Mark state # as accepting.
8. If M is incomplete (i.e., there are some (state, input) pairs for which no transition is defined), M requires a dead state. Add a new state D. For every (q, i) pair for which no transition has already been defined, create a transition from q to D labeled i. For every i in Σ, create a transition from D to D labeled i.

***FSM* → *Regular grammar*:** The construction is effectively the reverse of the one we just did. We leave this step as an exercise.

EXAMPLE 7.2 Strings that End with aaaa

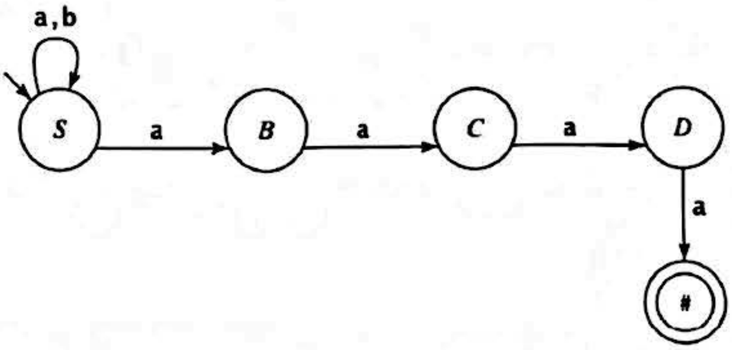
Let $L = \{w \in \{\text{a},\text{b}\}^*: w$ ends with the pattern aaaa$\}$. Alternatively, $L = (\text{a} \cup \text{b})^*$ aaaa. The following regular grammar defines L:

$S \rightarrow \text{a}S$	/* An arbitrary number of a's and b's can be generated
$S \rightarrow \text{b}S$	before the pattern starts.
$S \rightarrow \text{a}B$	/* Generate the first a of the pattern.

EXAMPLE 7.2 ***(Continued)***

$B \rightarrow \mathrm{a}C$	/* Generate the second a of the pattern.
$C \rightarrow \mathrm{a}D$	/* Generate the third a of the pattern.
$D \rightarrow \mathrm{a}$	/* Generate the last a of the pattern and quit.

Applying *grammartofsm* to this grammar, we get, omitting the dead state:



Notice that the machine that *grammartofsm* builds is not necessarily deterministic.

EXAMPLE 7.3 The Missing Letter Language

Let $\Sigma = \{\mathrm{a, b, c}\}$. Let L be $L_{Missing} = \{w :$ there is a symbol $a_i \in \Sigma$ not appearing in $w\}$, which we defined in Example 5.12. The following grammar G generates $L_{Missing}$:

$S \rightarrow \varepsilon$

$S \rightarrow \mathrm{a}B$

$S \rightarrow \mathrm{a}C$

$S \rightarrow \mathrm{b}A$

$S \rightarrow \mathrm{b}C$

$S \rightarrow \mathrm{c}A$

$S \rightarrow \mathrm{c}B$

$A \rightarrow \mathrm{b}A$

$A \rightarrow \mathrm{c}A$

$A \rightarrow \varepsilon$

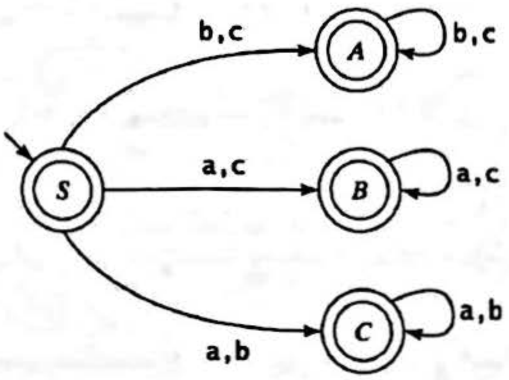
$B \rightarrow \mathrm{a}B$

$B \rightarrow \mathrm{c}B$

$B \rightarrow \varepsilon$
$C \rightarrow \text{a}C$
$C \rightarrow \text{b}C$
$C \rightarrow \varepsilon$

The job of S is to generate some string in $L_{Missing}$. It does that by choosing a first character of the string and then choosing which other character will be missing. The job of A is to generate all strings that do not contain any a's. The job of B is to generate all strings that do not contain any b's. And the job of C is to generate all strings that do not contain any c's.

If we apply *grammartofsm* to G, we get M =



M is identical to the NDFSM we had previously built for $L_{Missing}$ except that it waits to guess whether to go to A, B or C until it has seen its first input character.

Our proof of the first half of Theorem 7.1 clearly describes the correspondence between the nonterminals in a regular grammar and the states in a corresponding FSM. This correspondence suggests a natural way to think about the design of a regular grammar. The nonterminals in such a grammar need to "remember" the relevant state of a left-to-right analysis of a string.

EXAMPLE 7.4 Satisfying Multiple Criteria

Let $L = \{w \in \{\text{a}, \text{b}\}^*: w$ contains an odd number of a's and w ends in a$\}$. We can write a regular grammar G that defines L. G will contain four nonterminals, each with a unique function (corresponding to the states of a simple FSM that accepts L). So, in any derived string, if the remaining nonterminal is:

- S, then the number of a's so far is even. We don't have worry about whether the string ends in a since, to derive a string in L, it will be necessary to generate at least one more a anyway.

EXAMPLE 7.4 *(Continued)*

- T, then the number of a's so far is odd and the derived string ends in a.
- X, then the number of a's so far is odd and the derived string does not end in a.

Since only T captures the situation in which the number of a's so far is odd and the derived string ends in a, T is the only nonterminal that can generate ε. G contains the following rules:

$S \rightarrow \text{b}S$	/* Initial b's don't matter.
$S \rightarrow \text{a}T$	/* After this, the number of a's is odd and the generated string ends in a.
$T \rightarrow \varepsilon$	/* Since the number of a's is odd, and the string ends in a, it's okay to quit.
$T \rightarrow \text{a}S$	/* After this, the number of a's will be even again.
$T \rightarrow \text{b}X$	/* After this, the number of a's is still odd but the generated string no longer ends in a.
$X \rightarrow \text{a}S$	/* After this, the number of a's will be even.
$X \rightarrow \text{b}X$	/* After this, the number of a's is still odd and the generated string still does not end in a.

To see how this grammar works, we can watch it generate the string baaba:

$S \Rightarrow \text{b}S$	/* Still an even number of a's.
$\Rightarrow \text{ba}T$	/* Now an odd number of a's and ends in a. The process could quit now since the derived string, ba, is in L.
$\Rightarrow \text{baa}S$	/* Back to having an even number of a's, so it doesn't matter what the last character is.
$\Rightarrow \text{baab}S$	/* Still even a's.
$\Rightarrow \text{baaba}T$	/* Now an odd number of a's and ends in a. The process can quit, by applying the rule $T \rightarrow \varepsilon$.
$\Rightarrow \text{baaba}$	

So now we know that regular grammars define exactly the regular languages. But regular grammars are not often used in practice. The reason, though, is not that they couldn't be. It is simply that there is something better. Given some regular language L, the structure of a reasonable FSM for L very closely mirrors the structure of a reasonable regular grammar for it. And FSMs are easier to work with. In addition, there exist regular expressions. In Parts III and IV, as we move outward to larger classes of languages, there will no longer exist a technique like regular expressions.

At that point, particularly as we are considering the context-free languages, we will see that grammars are a very important and useful way to define languages.

Exercises

1. Show a regular grammar for each of the following languages:
   a. $\{w \in \{a, b\}^*$: w contains an even number of a's and an odd number of b's$\}$.
   b. $\{w \in \{a, b\}^*$: w does not end in aa$\}$.
   c. $\{w \in \{a, b\}^*$: w contains the substring abb$\}$.
   d. $\{w \in \{a, b\}^*$: if w contains the substring aa then $|w|$ is odd$\}$.
   e. $\{w \in \{a, b\}^*$: w does not contain the substring aabb$\}$.
2. Consider the following regular grammar G:

   $S \rightarrow aT$
   $T \rightarrow bT$
   $T \rightarrow a$
   $T \rightarrow aW$
   $W \rightarrow \varepsilon$
   $W \rightarrow aT$

   a. Write a regular expression that generates $L(G)$.
   b. Use *grammartofsm* to generate an FSM M that accepts $L(G)$.
3. Consider again the FSM M shown in Exercise 5.1. Show a regular grammar that generates $L(M)$.
4. Show by construction that, for every FSM M there exists a regular grammar G such that $L(G) = L(M)$.
5. Let $L = \{w \in \{a, b\}^*$: every a in w is immediately followed by at least one b$\}$.
   a. Write a regular expression that describes L.
   b. Write a regular grammar that generates L.
   c. Construct an FSM that accepts L.