

9.3. Regular Languages

9.3.1. Properties of Regular Languages. Recall that a regular language is the language associated to a regular grammar, i.e., a grammar $G = (N, T, P, \sigma)$ in which every production is of the form:

$$A \rightarrow a \quad \text{or} \quad A \rightarrow aB \quad \text{or} \quad A \rightarrow \lambda,$$

where $A, B \in N$, $a \in T$.

Regular languages over an alphabet T have the following properties (recall that λ = 'empty string', $\alpha\beta$ = 'concatenation of α and β ', α^n = ' α concatenated with itself n times'):

1. \emptyset , $\{\lambda\}$, and $\{a\}$ are regular languages for all $a \in T$.
2. If L_1 and L_2 are regular languages over T the following languages also are regular:

$$\begin{aligned} L_1 \cup L_2 &= \{\alpha \mid \alpha \in L_1 \text{ or } \alpha \in L_2\} \\ L_1 L_2 &= \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\} \\ L_1^* &= \{\alpha_1 \dots \alpha_n \mid \alpha_k \in L_1, n \in \mathbb{N}\}, \\ T^* - L_1 &= \{\alpha \in T^* \mid \alpha \notin L_1\}, \\ L_1 \cap L_2 &= \{\alpha \mid \alpha \in L_1 \text{ and } \alpha \in L_2\}. \end{aligned}$$

We justify the above claims about $L_1 \cup L_2$, $L_1 L_2$ and L_1^* as follows. We already know how to combine two grammars (see 9.2.4) L_1 and L_2 to obtain $L_1 \cup L_2$, $L_1 L_2$ and L_1^* , the only problem is that the rules given in section 9.2.4 do not have the form of a regular grammar, so we need to modify them slightly (we use the same notation as in section 9.2.4):

1. *Union Rule:* Instead of adding $\sigma \rightarrow \sigma_1$ and $\sigma \rightarrow \sigma_2$, add all productions of the form $\sigma \rightarrow RHS$, where RHS is the right hand side of some production $(\sigma_1 \rightarrow RHS) \in P_1$ or $(\sigma_2 \rightarrow RHS) \in P_2$.
2. *Product Rule:* Instead of adding $\sigma \rightarrow \sigma_1 \sigma_2$, use σ_1 as starting symbol and replace each production $(A \rightarrow a) \in P_1$ with $A \rightarrow a\sigma_2$ and $(A \rightarrow \lambda) \in P_1$ with $A \rightarrow \sigma_2$.
3. *Closure Rule:* Instead of adding $\sigma \rightarrow \sigma_1 \sigma$ and $\sigma \rightarrow \lambda$, use σ_1 as starting symbol, add $\sigma_1 \rightarrow \lambda$, and replace each production $(A \rightarrow a) \in P_1$ with $A \rightarrow a\sigma_1$ and $(A \rightarrow \lambda) \in P_1$ with $A \rightarrow \sigma_1$.

9.3.2. Regular Expressions. Regular languages can be characterized as languages defined by *regular expressions*. Given an alphabet T , a regular expression over T is defined recursively as follows:

1. \emptyset , λ , and a are regular expressions for all $a \in T$.
2. If R and S are regular expressions over T the following expressions are also regular: (R) , $R + S$, $R \cdot S$, R^* .

In order to use fewer parentheses we assign those operations the following hierarchy (from do first to do last): $*$, \cdot , $+$. We may omit the dot: $\alpha \cdot \beta = \alpha\beta$.

Next we define recursively the language associated to a given regular expression:

$$\begin{aligned}
 L(\emptyset) &= \emptyset, \\
 L(\lambda) &= \{\lambda\}, \\
 L(a) &= \{a\} && \text{for each } a \in T, \\
 L(R + S) &= L(R) \cup L(S), \\
 L(R \cdot S) &= L(R)L(S) && \text{(language product),} \\
 L(R^*) &= L(R)^* && \text{(language closure).}
 \end{aligned}$$

So, for instance, the expression a^*bb^* represents all strings of the form $a^n b^m$ with $n \geq 0$, $m > 0$, $a^*(b + c)$ is the set of strings consisting of any number of a 's followed by a b or a c , $a(a + b)^*b$ is the set of strings over $\{a, b\}$ than start with a and end with b , etc.

Another way of characterizing regular languages is as sets of strings recognized by finite-state automata, as we will see next. But first we need a generalization of the concept of finite-state automaton.

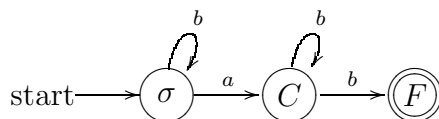
9.3.3. Nondeterministic Finite-State Automata. A *nondeterministic finite-state automaton* is a generalization of a finite-state automaton so that at each state there might be several possible choices for the “next state” instead of just one. Formally a nondeterministic finite-state automaton consists of

1. A finite set of *input symbols* \mathcal{I} .
2. A finite set of *states* \mathcal{S} .
3. A *next-state function* $f : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{P}(\mathcal{S})$.
4. A subset \mathcal{A} of \mathcal{S} of *accepting states*.

5. An *initial state* $\sigma \in \mathcal{S}$.

We represent the automaton $A = (\mathcal{I}, \mathcal{S}, f, \mathcal{A}, \sigma)$. We say that a nondeterministic finite-state automaton *accepts* a given string of input symbols if in its transition diagram there is a path from the starting state to an accepting state with its edges labeled by the symbols of the given string. A path (which we can express as a sequence of states) whose edges are labeled with the symbols of a string is said to *represent* the given string.

Example: Consider the nondeterministic finite-state automaton defined by the following transition diagram:



This automaton accepts precisely the strings of the form $b^n ab^m$, $n \geq 0$, $m > 0$. For instance the string $bbabb$ is represented by the path $(\sigma, \sigma, \sigma, C, C, F)$. Since that path ends in an accepting state, the string is accepted by the automaton.

Next we will see that there is a precise relation between regular grammars and nondeterministic finite-state automata.

Regular grammar associated to a nondeterministic finite-state automaton. Let A be a non-deterministic finite-state automaton given as a transition diagram. Let σ be the initial state. Let T be the set of inputs symbols and let N be the set of states. Let P be the set of productions

$$S \rightarrow xS'$$

if there is an edge labeled x from S to S' and

$$S \rightarrow \lambda$$

if S is an accepting state. Let G be the regular grammar

$$G = (N, T, P, \sigma).$$

Then the set of strings accepted by A is precisely $L(G)$.

Example: For the nondeterministic automaton defined above the corresponding grammar will be:

$T = \{a, b\}$, $N = \{\sigma, C, F\}$, with the productions

$$\sigma \rightarrow b\sigma, \quad \sigma \rightarrow aC, \quad C \rightarrow bC, \quad C \rightarrow bF, \quad F \rightarrow \lambda.$$

The string $bbabb$ can be produced like this:

$$\sigma \Rightarrow b\sigma \Rightarrow bb\sigma \Rightarrow bbaC \Rightarrow bbabC \Rightarrow bbabbF \Rightarrow bbabb.$$

Nondeterministic finite-state automaton associated to a given regular grammar. Let $G = (N, T, P, \sigma)$ be a regular grammar. Let

$$\mathcal{I} = T$$

$$\mathcal{S} = N \cup \{F\}, \text{ where } F \notin N \cup T$$

$$f(S, x) = \{S' \mid S \rightarrow xS' \in P\} \cup \{F \mid S \rightarrow x \in P\}$$

$$\mathcal{A} = \{F\} \cup \{S \mid S \rightarrow \lambda \in P\}.$$

Then the nondeterministic finite-state automaton $\mathcal{A} = (\mathcal{I}, \mathcal{S}, f, \mathcal{A}, \sigma)$ accepts precisely the strings in $L(G)$.

9.3.4. Relationships Between Regular Languages and Automata. In the previous section we saw that regular languages coincide with the languages accepted by nondeterministic finite-state automata. Here we will see that the term “nondeterministic” can be dropped, so that regular languages are precisely those accepted by (deterministic) finite-state automata. The idea is to show that given any nondeterministic finite-state automata it is possible to construct an equivalent deterministic finite-state automata accepting exactly the same set of strings. The main result is the following:

Let $A = (\mathcal{I}, \mathcal{S}, f, \mathcal{A}, \sigma)$ be a nondeterministic finite-state automaton. Then A is equivalent to the finite-state automaton $A' = (\mathcal{I}', \mathcal{S}', f', \mathcal{A}', \sigma')$, where

1. $\mathcal{S}' = \mathcal{P}(\mathcal{S})$.
2. $\mathcal{I}' = \mathcal{I}$.
3. $\sigma' = \{\sigma\}$.
4. $\mathcal{A}' = \{X \subseteq \mathcal{S} \mid X \cap \mathcal{A} \neq \emptyset\}$.
5. $f'(X, x) = \bigcup_{S \in X} f(S, x), \quad f'(\emptyset, x) = \emptyset.$

```

graph LR
    start((start)) --> sigma((sigma))
    sigma -- b --> sigma
    sigma -- a --> C((C))
    C -- b --> C
    C -- b --> F(((F)))
    style start fill:none,stroke:none
    style F fill:none,stroke:none
  
```

$$\mathcal{S}' = \{\emptyset, \{\sigma\}, \{C\}, \{F\}, \{\sigma, C\}, \{\sigma, F\}, \{C, F\}, \{\sigma, C, F\}\}.$$
$$\mathcal{A}' = \{\{F\}, \{\sigma, F\}, \{C, F\}, \{\sigma, C, F\}\}.$$

```

graph LR
    start((start)) --> S1({σ})
    S1 -- b --> S1
    S1 -- a --> S2({C})
    S2 -- b --> S2
    S2 -- b --> S3({C, F})
    S3 -- b --> S3
    S3 -- a --> S4(∅)
    S4 -- a --> S4
    S4 -- a --> S3
  
```

So, once proved that every nondeterministic finite-state automaton is equivalent to some deterministic finite-state automaton, we obtain the main result of this section: A language L is regular if and only if there exists a finite-state automaton that accepts precisely the strings in L .