10.3. Language Recognition

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

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

where $A, B \in N = V - T$, $a \in T$.

Regular languages over an alphabet T have the following properties (recall that $\lambda =$ 'empty string', $\alpha\beta =$ 'concatenation of α and β', $\alpha^n =$ '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_1L_2 and L_1^* as follows. We already know how to combine two grammars (see 10.2.4) L_1 and L_2 to obtain $L_1 \cup L_2$, L_1L_2 and L_1^*, the only problem is that the rules given in section 10.2.4 do no have the form of a regular grammar, so we need to modify them slightly (we use the same notation as in section 10.2.4):

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

10.3.2. 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.

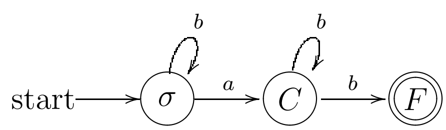
10.3.3. Nondeterministic Finite-State Automata. A *nondeterministic finite-state automaton* is a generalization of a finite-state automaton so that at each state there might be several possible choices for the "next state" instead of just one. Formally a nondeterministic finite-state automaton consists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

1. A finite set of *states* $\mathcal{S}$.
2. A finite set of *input symbols* $\mathcal{I}$.
3. A *next-state* or *transition function* $f : \mathcal{S} \times \mathcal{I} \to \mathcal{P}(\mathcal{S})$.
4. An *initial state* $\sigma \in \mathcal{S}$.

5. A subset $\mathcal{F}$ of $\mathcal{S}$ of *accepting* or *final states*.

We represent the automaton $A = (\mathcal{S}, \mathcal{I}, f, \sigma, \mathcal{F})$. We say that a nondeterministic finite-state automaton *accepts* or *recognizes* a given string of input symbols if in its transition diagram there is a path from the starting state to a final 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 recognizes 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 a final state, the string is recognized 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, let N be the set of states, and $V = N \cup T$. Let P be the set of productions

$$S \to xS'$$

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

$$S \to \lambda$$

if S is a final state. Let G be the regular grammar

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

Then the set of strings recognized 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 productions

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

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 $G = (V, T, \sigma, P)$ be a regular grammar. Let

$\mathcal{I} = T$.

$\mathcal{S} = N \cup \{F\}$, where $N = V - T$, and $F \notin V$.

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

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

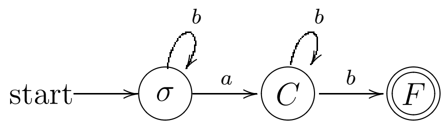
Then the nondeterministic finite-state automaton $A = (\mathcal{S}, \mathcal{I}, f, \sigma, \mathcal{F})$ recognizes precisely the strings in $L(G)$.

10.3.4. Relationships Between Regular Languages and Automata. In the previous section we saw that regular languages coincide with the languages recognized by nondeterministic finite-state automata. Here we will see that the term “nondeterministic” can be dropped, so that regular languages are precisely those recognized 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 recognizing exactly the same set of strings. The main result is the following:

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

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

Example: Find a (deterministic) finite-state automaton A' equivalent to the following nondeterministic finite-state automaton A:



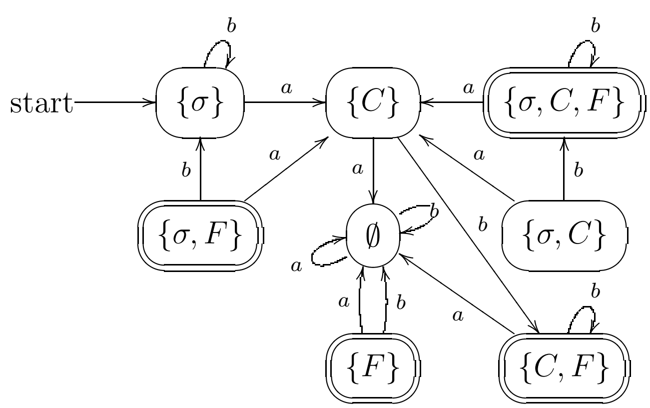
Answer: The set of input symbols is the same as that of the given automaton: $\mathcal{I}' = \mathcal{I} = \{a, b\}$. The set of states is the set of subsets of $\mathcal{S} = \{\sigma, C, F\}$, i.e.:

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

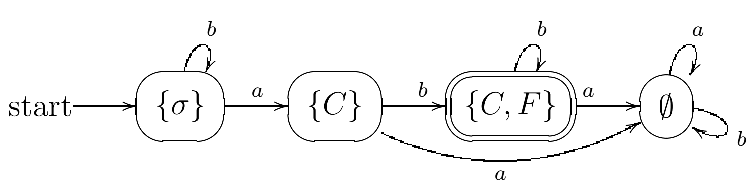
The starting state is $\{\sigma\}$. The final states of A' are the elements of $\mathcal{S}'$ containing some final state of A:

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

Then for each element X of $\mathcal{S}'$ we draw an edge labeled x from X to $\displaystyle\bigcup_{S \in X} f(S, x)$ (and from $\emptyset$ to $\emptyset$):



We notice that some states are unreachable from the starting state. After removing the unreachable states we get the following simplified version of the finite-state automaton:



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 recognizes precisely the strings in L.