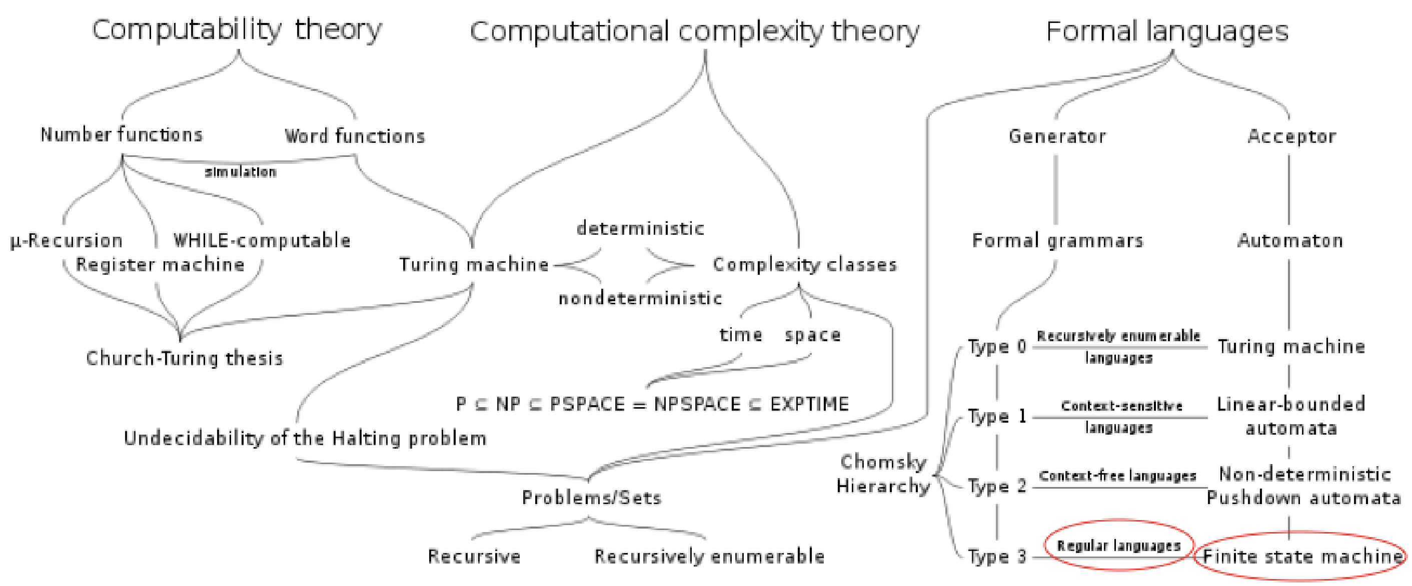
Generators vs. Recognizers

Up to now we have only described languages in terms of machines that **recognize** a particular language.

But we could also imagine describing a language by a system that is able to **generate** all the strings in a language.



Type 3: Regular Expressions

Rewriting Systems

In order to define a system that generates a language we introduce a new model of computation: Rewriting Systems.

Informally, a rewriting system consists of an alphabet and a set of rules over that alphabet.

You are already familiar with a very powerful rewriting system: Algebra!

Here, the alphabet are the numerals and variable names in addition to operator names. The rules consist of your standard algebraic laws.

Rewriting Systems

Example: Consider the set of algebraic laws:

$$x + x = 2 \times x \quad (1)$$
$$y + 0 = y \quad (2)$$
$$x + y = y + x \quad (3)$$

We can apply these rules to strings formed from the alphabet. Consider:

$$\begin{aligned} 5+3+5+0 &= 5+3+5 && \text{(rule 2)} \\ &= 5+5+3 && \text{(rule 3)} \\ &= 2 \times 5 + 3 && \text{(rule 1)} \end{aligned}$$

The string that we start with is called the **input string** and the string that we end up with is called the **normal form** because no other rules apply to this final string.

For our purposes we introduce a special rewriting system called a **String Rewriting System.**

String Rewriting Systems

Definition: [String Rewriting System (SRS)] A *string rewriting system* is a tuple (Σ, R) where,

- Σ is a finite *alphabet* where Σ^* is the set of (possibly empty) strings over Σ.[a]
- R is a binary relation on Σ^*, i.e., $R \subseteq \Sigma^* \times \Sigma^*$. Each element $(u, v) \in R$ is called a rewriting rule and is usually written as $u \to v$.

An inference step in this formal system is: given a string u and a rule $u \to v$ with $u, v \in \Sigma^*$ and $u \to v \in R$ then the string u can be *rewritten* as the string v.

[a] The set Σ^* is a convenient short hand to describe all the strings over the alphabet Σ.

String Rewriting Systems

In order for an SRS (Σ, R) to be useful we allow rules to be applied to substrings of given strings; let $s = xuy$,$t = xvy$, and $u \rightarrow v \in R$ with $x, y, u, v \in \Sigma^*$, then we say that s *rewrites to* t and we write,

$$s \Rightarrow t.$$

More formally,

Definition: [one-step rewriting relation] Let (Σ, R) be a string rewriting system, then the *one-step rewriting relation* RW is defined as the set $\Sigma^* \times \Sigma^*$ with $s \Rightarrow t \in RW$ for strings $s, t \in \Sigma^*$ if and only if there exist $x, y, u, v \in \Sigma^*$ such that $s = xuy$, $t = xvy$, and $u \longrightarrow v \in R$.

In plain English: any two string s, t belong to the relation RW if and only if they can be related by a rewrite rule in the rule set R.

Exercise: $R \subseteq RW$. Why? (spoiler alert, next page holds the solution)

String Rewriting Systems

Proposition: $R \subseteq RW$.

Proof: We use the definition of a subset, $R \subseteq RW$ iff $\forall e \in R.\, e \in RW$, for our proof. There is nothing to prove for the 'only if' direction. More interesting is the 'if' direction, if we can show that all elements of R are also elements of RW then it follows from the definition that $R \subseteq RW$.

An element of R is the pair (u, v) with $u, v \in \Sigma^*$ if the rewriting system contains the rule $u \longrightarrow v$. An element of RW is the pair (xuy, xvy) with $u, v, x, y \in \Sigma^*$ if the rewriting system contains the rule $u \longrightarrow v$. Thus, RW contains pairs of strings where the first string contains a substring that is the left side of a rule in the rewriting system. Observe that $(u, v) \in RW$ with x and y the empty strings. It follows that all elements of R are members of RW.$\square$

String Rewriting Systems

Given a string rewriting system (Σ, R), we can obviously apply the rewriting rules to the results of a rewriting step. This gives rise to *derivations*

$$s_n \Rightarrow s_{n-1} \Rightarrow \ldots \Rightarrow s_1 \Rightarrow s_0,$$

with $s_k \in \Sigma^*$.

We say that s_0 is a *normal form* if s_0 cannot be rewritten any further.

The *transitive closure* $\Rightarrow^*$ of the one-step rewriting relation is the set all pairs of strings that are related to each other via zero or more rewriting steps, e.g.,

$$s_n \Rightarrow^* s_0,$$

and

$$s_i \Rightarrow^* s_i.$$

String Rewriting Systems

Example: The urn game. An urn contains black and white beads. The game has the following rules:

- if you remove two black beads you have to replace them with a black bead.
- if you remove two white beads you have to replace them with a black bead.
- if you remove a white and a black bead you have to replace them with a white bead.

Given the contents of an urn, what is the outcome of the game?

The game can be set up as a string rewriting system (Σ, R). Let $\Sigma = \{\text{black}, \text{white}\}$ and let R be the following set of rules,

$$
\begin{array}{rcl}
\text{black black} & \longrightarrow & \text{black} \\
\text{white white} & \longrightarrow & \text{black} \\
\text{black white} & \longrightarrow & \text{white} \\
\text{white black} & \longrightarrow & \text{white}
\end{array}
$$

black white black white $\Rightarrow$ black white white $\Rightarrow$ white white $\Rightarrow$ black

black black white white $\Rightarrow$ black white white $\Rightarrow$ white white $\Rightarrow$ black

black black white $\Rightarrow$ black white $\Rightarrow$ white

black white black $\Rightarrow$ black white $\Rightarrow$ white

String Rewriting Systems

Observations:

- It can be shown that for each urn there exists a unique normal form, the order of rule application does not matter.
- If we interpret a rewrite rule $u \to v$ as specifying that u *is the same as* v then we can interpret the normal form as a 'value' for an urn. Consider,

$$\text{black white black} \Rightarrow \text{black white} \Rightarrow \text{white},$$

  the normal form 'white' can be considered the value for the urn.
- We say that two urns are equivalent if they have the same normal form,

$$\begin{tikzcd} \text{black white black} \arrow[dr, Rightarrow, "*"] & \equiv & \text{black black white} \arrow[dl, Rightarrow, "*"'] \\ & \text{white} & \end{tikzcd}$$

String Rewriting Systems

Example: Palindrome generator. We construct a string rewriting system (Σ, R) with $\Sigma = \{a, b, \ldots, z, \alpha\}$ and R the set of rules,

$$\begin{array}{rcl} \alpha & \longrightarrow & a\alpha a \\ \alpha & \longrightarrow & b\alpha b \\ & \vdots & \\ \alpha & \longrightarrow & z\alpha z \\ a\alpha a & \longrightarrow & a \\ b\alpha b & \longrightarrow & b \\ & \vdots & \\ z\alpha z & \longrightarrow & z \\ \alpha & \longrightarrow & \epsilon \end{array}$$

$$\alpha \Rightarrow r\alpha r \Rightarrow ra\alpha ar \Rightarrow rad\alpha dar \Rightarrow radar$$

Exercise: Derive the normal form: $racecar$

Exercise: Derive the normal form: $redder$

Grammars

Observations:

- We have seen in the case of the palindrome generator that SRSs are well suited for generating strings with structure.
- By modifying the standard SRS just slightly we obtain a convenient framework for generating strings with desirable structure – *Grammars*

Definition: [Grammar] A *grammar* is a 4-tuple (V, Σ, R, s) such that,

- V is a set of variables called the *non-terminals*,
- Σ with $V \cap \Sigma = \emptyset$, is a set of symbols called the *terminals*,[a]
- R is a set of rules of the form $u \longrightarrow v$ with $u, v \in (V \cup \Sigma)^*$,[b]
- s is called the *start symbol* and $s \in V$.

[a] The fact that V and Σ are non-overlapping means that there will never be confusion between terminals and non-terminals.

[b] All sets in this definition are considered to be *finite*.

Grammars

Example: Grammar for arithmetic expressions. We define the grammar (V, Σ, R, s) as follows:

- $V = \{E\}$,
- $\Sigma = \{a, b, c, +, *, (,)\}$,
- R is the set of rules,

$$\begin{array}{lcl} E & \longrightarrow & E + E \\ E & \longrightarrow & E * E \\ E & \longrightarrow & (E) \\ E & \longrightarrow & a \\ E & \longrightarrow & b \\ E & \longrightarrow & c \end{array}$$

- $s = E$ (clearly this satisfies $s \in V$).

With grammars,derivations always start with the start symbol. Consider,

$$E \Rightarrow E * E \Rightarrow (E) * E \Rightarrow (E + E) * E \Rightarrow (a + E) * E \Rightarrow (a + b) * E \Rightarrow (a + b) * c.$$

Here, $(a + b) * c$ is a normal form often also called a *terminal* or *derived string*.

Grammars

Exercise: Identify the rule that was applied at each rewrite step in the above derivation.

Exercise: Derive the string $((a))$.

Exercise: Derive the string $a + b * c$.

Grammars

Example: Grammar for strings of a's and b's with at least one b in them. We define the grammar (V, Σ, R, s) as follows:

- $V = \{S, A, B\}$,
- $\Sigma = \{a, b\}$,
- R is the set of rules,

$$
\begin{array}{ccl}
S & \longrightarrow & A\,b\,B \\
A & \longrightarrow & \epsilon \\
A & \longrightarrow & a\ A \\
A & \longrightarrow & b\ A \\
B & \longrightarrow & \epsilon \\
B & \longrightarrow & a\ B \\
B & \longrightarrow & b\ B
\end{array}
$$

- $s = S$.

Exercise: Derive string aba.

Exercise: Derive string bbb.

Exercise: Derive string b.

Grammars

We are now in the position to define exactly what we mean by the *language of a grammar*.

Definition:[Language of a Grammar] Let $G = (V, \Sigma, R, s)$ be a grammar, then we define the *language of grammar* G as the set of all terminal strings that can be derived from the start symbol s by rewriting using the rules in R. Formally,

$$L(G) = \{q \mid s \Rightarrow^* q \wedge q \in \Sigma^*\}.$$

Example: Let $J = (V, \Sigma, R, s)$ be the grammar of Java, then $L(J)$ is the set of all possible Java programs.

Grammars

Observations:

- With the concept of a language we can now ask interesting questions. For example, given a grammar $G = (V, \Sigma, R, s)$ and some sentence $p \in \Sigma^*$, does p belong to $L(G)$?
- If we let J be the grammar of Java, then asking whether some string $p \in \Sigma^*$ is in $L(J)$ is equivalent to asking whether p is a *syntactically correct program*.
- We can prove language membership by by showing that the sentence p in question can be derived from the start symbol. Graphically,

$$
\begin{tikzcd}
s \arrow[dr, Rightarrow, "*"] & \equiv & p \arrow[dl, Rightarrow, "*"'] \\
& p &
\end{tikzcd}
$$

Grammars

Observations:

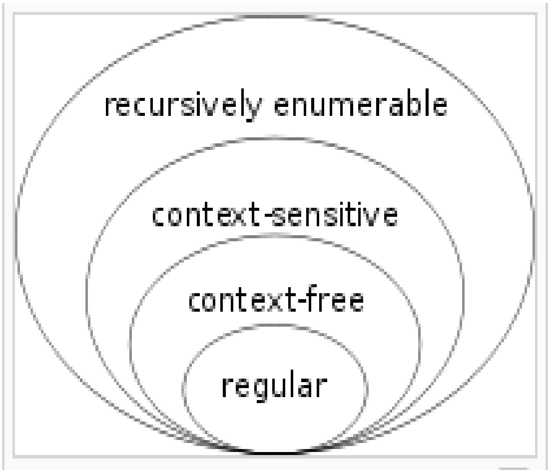
- By restricting the shape of the rewrite rules in a grammar we obtain different language *classes*.
- The most famous set of language classes is the *Chomsky Hierarchy*.

The Chomsky Hierarchy

Let $G = (V, \Sigma, R, s)$ be a grammar. Restricting the shape of the rules in R gives rise to the following hierarchy.

Rules	Grammar	Language	Machine
$\alpha \longrightarrow \beta$	Type-0	Recursively Enumerable	Turing machine
$\alpha A \beta \longrightarrow \alpha\gamma\beta$	Type-1	Context-sensitive	Linear-bounded Turing machine
$A \longrightarrow \gamma$	Type-2	Context-free	Pushdown automaton
$A \longrightarrow a$ and $A \longrightarrow aB$	Type-3	Regular	Finite state automaton

where $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$, $A, B \in V$, $a \in \Sigma$. In Type-1 γ is not allowed to be the empty string.



Type 3: Regular Grammars

A grammar $G = (V, \Sigma, R, s)$ is called regular (type 3) if and only if the rules in R are of the form [a]

$$A \longrightarrow a\,B$$

or

$$A \longrightarrow a$$

with $A, B \in V$ and $a \in \Sigma$.

---

[a] If the language include the empty string then the rule $s \longrightarrow \epsilon$ will need to be added to the grammar.

Type 3: Regular Grammars

Example: Grammar for strings of one or more 1's followed by a single 0. We define the grammar (V, Σ, R, s) as follows:

- $V = \{A, S\}$,
- $\Sigma = \{0, 1\}$,
- R is the set of rules,

$$\begin{array}{lcl} S & \longrightarrow & 1\ A \\ A & \longrightarrow & 1\ A \\ A & \longrightarrow & 0 \end{array}$$

- $s = S$.

Type 3: Regular Grammars

Example: Grammar for strings of a's and b's with at least one b in them. We define the grammar (V, Σ, R, s) as follows:

- $V = \{A, B\}$,
- $\Sigma = \{a, b\}$,
- R is the set of rules,

$$\begin{array}{ccc} A & \longrightarrow & a\ A \\ A & \longrightarrow & b\ A \\ A & \longrightarrow & b\ B \\ A & \longrightarrow & b \\ B & \longrightarrow & a\ B \\ B & \longrightarrow & b\ B \\ B & \longrightarrow & a \\ B & \longrightarrow & b \end{array}$$

- $s = A$.

This shows that the language of strings of a's and b's with at least one b in them is a regular language.

Regular Languages and Regular Grammars

> **Lemma:** If a language is recognized by a FA then it is generated by a type-3 grammar.

Proof: We show that if a language is recognized by a DFA then we can construct a type-3 grammar that generates it. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes language $L(M)$. We now construct the type-3 grammar $G = (V, \Sigma, R, s)$ that simulates the computations of the DFA :

- For each state $q \in Q$ we construct the non-terminal symbol $\langle q \rangle \in V$,
- The terminal set Σ in the grammar is the same as the alphabet of the machine,
- We construct the rule set R as follows, let $q, p \in Q$ and let $a \in \Sigma$,
  - add a rule of the form $\langle q \rangle \rightarrow a \langle p \rangle$ for each transition $\delta(q, a) = p$,
  - add a rule of the form $\langle q \rangle \rightarrow a$ for each transition $\delta(q, a) = p$ where $p \in F$,
  - add a rule of the form $\langle q_0 \rangle \rightarrow \epsilon$ if the initial state is an accepting state, i.e., $q_0 \in F$.
- We let $s = \langle q_0 \rangle$.

Regular Languages and Regular Grammars

Now,for any string $w = w_1 w_2 \ldots w_n \in L(M)$ the machine M will perform the computation

$$q_0 w_1 w_2 \ldots w_n \vdash w_1 q_1 w_2 \ldots w_n \vdash \ldots \vdash w_1 w_2 \ldots q_{n-1} w_n \vdash w_1 w_2 \ldots w_n q_n$$

with $q_n \in F$. We can show by induction on n that the input string is generated by the grammar with the derivation

$$\langle q_0 \rangle \Rightarrow w_1 \langle q_1 \rangle \Rightarrow w_1 w_2 \langle q_2 \rangle \Rightarrow \ldots \Rightarrow w_1 w_2 \ldots w_{n-1} \langle q_{n-1} \rangle \Rightarrow w_1 w_2 \ldots w_{n-1} w_n$$

Regular Languages and Regular Grammars

Consider:

1. $s = \epsilon$ – in the machine this gives rise to the computation q_0 which is also an accepting state, the grammar derives the empty string via the rule $\langle q_0 \rangle \longrightarrow \epsilon$.
2. $s = w_1$ – this gives rise to the computation $q_0 w_1 \vdash w_1 q_1$ where q_1 is an accepting state; the grammar derives string w_1 via the rule $\langle q_0 \rangle \longrightarrow w_1$.
3. Any substring $s = w_1 w_2 \ldots w_k$ of string $w = w_1 w_2 \ldots w_n \in L(M)$ with $k \leq n$– then the machine performs the computation

$$q_0 w_1 w_2 \ldots w_n \vdash w_1 q_1 w_2 \ldots w_n \vdash \ldots \vdash w_1 w_2 \ldots q_{k-1} w_k \vdash w_1 w_2 \ldots w_k q_k$$

where q_k might or might not be an accepting state; as inductive hypothesis we assume that the grammar derives the string $w_1 w_2 \ldots w_{k-1}$ with the following derivation

$$\langle q_0 \rangle \Rightarrow w_1 \langle q_1 \rangle \Rightarrow w_1 w_2 \langle q_2 \rangle \Rightarrow \ldots \Rightarrow w_1 w_2 \ldots w_{k-1} \langle q_{k-1} \rangle$$

then it follows from the inductive hypothesis and the fact that by construction there has to exist at least one of the following rules

$$\langle q_{k-1} \rangle \longrightarrow w_k$$

if q_k is an accepting state or

$$\langle q_{k-1} \rangle \longrightarrow w_k \langle q_k \rangle$$

if not, that the grammar can generate the string $s = w_1 w_2 \ldots w_k$.

□

Regular Languages and Regular Grammars

Lemma: if a language is generated by a type-3 grammar then it is recognized by a FA.

Proof: We show that if a language is generated by a type-3 grammar then it is recognized by a DFA. Let $G = (V, \Sigma, R, s)$ be a type-3 grammar, then we construct the machine $M = (Q, \Sigma, \delta, q_0, F)$ as follows,

- For each $A \in V$ in grammar G we construct the state $q_A \in Q$ in machine M,
- The terminal set Σ in G becomes the alphabet Σ for the machine,
- Construct the transition function δ as follows,
  - for each rule of the form $A \rightarrow a\,B \in R$ we construct the transition $\delta(q_A, a) = q_B$,
  - for each rule of the form $A \rightarrow a \in R$ we construct the transition $\delta(q_A, a) = q_F$ with $q_F \in F$,
  - for each rule of the form $A \rightarrow \epsilon \in R$ we add the state q_A to the set of accepting states, F.
- the initial state $q_s = q_0$.

Regular Languages and Regular Grammars

Now,for any string $w = w_1 w_2 \ldots w_n \in L(G)$, we can show by induction that a derivation in G,

$$\langle q_0 \rangle \Rightarrow w_1 \langle q_1 \rangle \Rightarrow w_1 w_2 \langle q_2 \rangle \Rightarrow \ldots \Rightarrow w_1 w_2 \ldots w_{n-1} \langle q_{n-1} \rangle \Rightarrow w_1 w_2 \ldots w_{n-1} w_n$$

has an equivalent computation for the machine M, the machine M will perform the computation,

$$q_0 w_1 w_2 \ldots w_n \vdash w_1 q_1 w_2 \ldots w_n \vdash \ldots \vdash w_1 w_2 \ldots q_{n-1} w_n \vdash w_1 w_2 \ldots w_n q_n$$

with $q_n \in F$. $\Box$.

Regular Languages and Regular Grammars

Theorem: A language is recognized by a FA if and only if it is generated by a type-3 grammar.

Proof: Follows directly from the two previous lemmas.

Regular Expressions

As you might have noticed, regular grammars are a little awkward to construct. There is a another generator for regular languages called *regular expressions*.

Regular Expressions

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

In items 1 and 2, the regular expressions a and ε represent the languages $\{a\}$ and $\{\varepsilon\}$, respectively. In item 3, the regular expression $\emptyset$ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2, or the star of the language R_1, respectively.

Regular Expressions

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
   The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
11. $1^*\emptyset = \emptyset$.
    Concatenating the empty set to any set yields the empty set.
12. $\emptyset^* = \{\varepsilon\}$.
    The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

Regular Languages and Regular Expressions

Theorem: A language is regular if and only if a regular expression generates it.

Proof Sketch:[a] Let L be some language.

If L is regular, then a regular expression generates it. If L is regular then some FA recognizes it. For every FA we can construct an equivalent regular expression.

If some regular expression generates L, then it is a regular language. For every regular expression that generates L we can construct an equivalent FA that recognizes L.

[a] A formal proof of this appears in the book; pp66ff 1st & 2nd eds.

Regular Grammars and Expressions

Corollary: Regular Grammars and Regular Expressions generate the same class of languages.

Follows immediately from the previous two theorems.