Introduction to the Theory of Computation
Languages, Automata, Grammars
Slides for CIS262

Jean Gallier

February 21, 2020

Chapter 1

Introduction

1.1 Generalities, Motivations, Problems

In this part of the course we want to understand

- What is a language?
- How do we define a language?
- How do we manipulate languages, combine them?
- What is the complexity of a language?

Roughly, there are two dual views of languages:

(A) The *recognition* point view.

(B) The *generation* point of view.

No matter how we view a language, we are typically considering two things:

(1) The *syntax*, i.e., what are the "legal" strings in that language (what are the "grammar rules"?).

(2) The *semantics* of strings in the language, i.e., what is the *meaning* (or *interpretation*) of a string.

The semantics is usually a lot more interesting than the syntax but unfortunately much more difficult to deal with!

Therefore, sorry, we will only be dealing with syntax!

In (A), we typically assume some kind of "black box", M, (an *automaton*) that takes a string, w, as input and returns two possible answers:

Yes, the string w is *accepted*, which means that w belongs to the language, L, that we are trying to define.

No, the string w is *rejected*, which means that w *does not* belong to the language, L.

Usually, the black box M gives a definite answer for every input after a finite number of steps, but not always.

For example, a Turing machine may go on computing forever and not give any answer for certain strings not in the language. This is an example of *undecidability*.

The black box may compute *deterministically* or *non-deterministically*, which means roughly that on input w, the machine M is allowed to try different computations and to ignore failing computations as long as there is some successful computation on input w.

This affects greatly the *complexity* of recognition, i.e,. how many steps it takes to process w.

Sometimes, a nondeterministic version of an automaton turns out to be equivalent to the deterministic version (although, with different complexity).

This tends to happen for very restrictive models—where nondeterminism does not help, or for very powerful models—where again, nondeterminism does not help, but because the deterministic model is already very powerful!

We will investigate automata of increasing power of recognition:

(1) Deterministic and nondeterministic finite automata (DFA's and NFA's, their power is the same).

(2) Pushdown automata (PDA's) and determinstic pushdown automata (DPDA's), here PDA $>$ DPDA.

(3) Deterministic and nondeterministic Turing machines (their power is the same).

(4) If time permits, we will also consider some restricted type of Turing machine known as LBA (linear bounded automaton).

In (B), we are interested in formalisms that specify a language in terms of *rules* that allow the generation of "legal" strings. The most common formalism is that of a formal *grammar*.

Remember:

- An automaton *recognizes* (or *accepts*) a language,
- a grammar *generates* a language.
- gramm*a*r is spelled with an "a" (not with an "e").
- The plural of automat*on* is automat*a* (not automat*ons*).

For "good" classes of grammars, it is possible to build an automaton, M_G, from the grammar, G, in the class, so that M_G recognizes the language, $L(G)$, generated by the grammar G.

However, grammars are nondeterministic in nature. Thus, even if we try to avoid nondeterministic automata, we usually can't escape having to deal with them.

We will investigate the following types of grammars (the so-called *Chomsky hierarchy*) and the corresponding families of languages:

(1) Regular grammars (type 3-languages).

(2) Context-free grammars (type 2-languages).

(3) The recursively enumerable languages or r.e. sets (type 0-languages).

(4) If time permit, context-sensitive languages (type 1-languages).

Miracle: The grammars of type (1), (2), (3), (4) correspond exactly to the automata of the corresponding type!

Furthermore, there are *algorithms* for converting grammars to the corresponding automata (and backward), although some of these algorithms are not practical.

Building an automaton from a grammar is an important practical problem in language processing. A lot is known for the regular and the context-free grammars, but there is still room for improvements and innovations!

There are other ways of defining families of languages, for example

Inductive closures.

In this style of definition, a collection of basic (atomic) languages is specified, some operations to combine languages are also specified, and the family of languages is defined as the smallest one containing the given atomic languages and closed under the operations.

Investigating closure properties (for example, union, intersection) is a way to assess how “robust” (or complex) a family of languages is.

Well, it is now time to be precise!

Chapter 2

Basics of Formal Language Theory

2.1 Review of Some Basic Math Notation and Definitions

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$.

The *natural numbers*,

$$\mathbb{N} = \{0, 1, 2, \ldots\}.$$

The *integers*,

$$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}.$$

The *rationals*,

$$\mathbb{Q} = \left\{ \frac{p}{q} \mid p, q \in \mathbb{Z},\ q \neq 0 \right\}.$$

The *reals*, $\mathbb{R}$.

The *complex numbers*,

$$\mathbb{C} = \{a + ib \mid a, b \in \mathbb{R}\}.$$

Given any set X, the *power set* of X is the set of all subsets of X and is denoted 2^X.

The notation

$$f\colon X \to Y$$

denotes a *function* with *domain X* and *range* (or *codomain*) Y.

$$\mathrm{graph}(f) = \{(x, f(x)) \mid x \in X\} \subseteq X \times Y$$

is the *graph* of f.

$$\mathrm{Im}(f) = f(X) = \{y \in Y \mid \exists x \in X,\, y = f(x)\} \subseteq Y$$

is the *image* of f.

More generally, if $A \subseteq X$, then

$$f(A) = \{y \in Y \mid \exists x \in A,\, y = f(x)\} \subseteq Y$$

is the *(direct) image* of A.

If $B \subseteq Y$, then

$$f^{-1}(B) = \{x \in X \mid f(x) \in B\} \subseteq X$$

is the *inverse image* (or *pullback*) of B.

$f^{-1}(B)$ is a set; it might be empty even if $B \neq \emptyset$.

Given two functions $f\colon X \to Y$ and $g\colon Y \to Z$, the function $g \circ f\colon X \to Z$ given by

$$(g \circ f)(x) = g(f(x)) \quad \text{for all } x \in X$$

is the *composition* of f and g.

The function $\mathrm{id}_X\colon X \to X$ given by

$$\mathrm{id}_X(x) = x \quad \text{for all } x \in X$$

is the *identity function* (of X).

A function $f\colon X \to Y$ is *injective* (old terminology *one-to-one)* if for all $x_1, x_2 \in X$,
if $f(x_1) = f(x_2)$, then $x_1 = x_2$;

equivalently if $x_1 \neq x_2$, then $f(x_1) \neq f(x_2)$.

Fact: If $X \neq \emptyset$ (and so $Y \neq \emptyset$), a function $f\colon X \to Y$ is injective iff there is a function $r\colon Y \to X$ (a *left inverse*) such that

$$r \circ f = \mathrm{id}_X.$$

Note: r is surjective.

A function $f\colon X \to Y$ is *surjective* (old terminology *onto)* if for all $y \in Y$, there is some $x \in X$ such that $y = f(x)$, iff

$$f(X) = Y.$$

Fact: If $X \neq \emptyset$ (and so $Y \neq \emptyset$), a function $f\colon X \to Y$ is surjective iff there is a function $s\colon Y \to X$ (a *right inverse* or *section*) such that

$$f \circ s = \mathrm{id}_Y.$$

Note: s is injective.

A function $f\colon X \to Y$ is *bijective* if it is injective and surjective.

Fact: If $X \neq \emptyset$ (and so $Y \neq \emptyset$), a function $f\colon X \to Y$ is bijective if there is a function $f^{-1}\colon Y \to X$ which is a left and a right inverse, that is

$$f^{-1} \circ f = \mathrm{id}_X, \qquad f \circ f^{-1} = \mathrm{id}_Y.$$

The function f^{-1} is unique and called the *inverse* of f. The function f is said to be *invertible*.

A *binary relation* R between two sets X and Y is a subset

$$R \subseteq X \times Y = \{(x, y) \mid x \in X,\, y \in Y\}.$$

$$\mathrm{dom}(R) = \{x \in X \mid \exists y \in Y,\, (x, y) \in R\} \subseteq X$$

is the *domain* of R.

$$\mathrm{range}(R) = \{y \in Y \mid \exists x \in X,\, (x, y) \in R\} \subseteq Y$$

is the *range* of R.

We also write xRy instead of $(x, y) \in R$.

Given two relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$, their *composition* $R \circ S \subseteq X \times Z$ is given by

$$R \circ S = \{(x, z) \mid \exists y \in Y, (x, y) \in R \quad \text{and} \quad (y, z) \in S\}.$$

Note that if R and S are the graphs of two functions f and g, then $R \circ S$ is the graph of $g \circ f$.

$$I_X = \{(x, x) \mid x \in X\}$$

is the *identity relation on X*.

Given $R \subseteq X \times Y$, the *converse* $R^{-1} \subseteq Y \times X$ of R is given by

$$R^{-1} = \{(x, y) \in Y \times X \mid (y, x) \in R\}.$$

A relation $R \subseteq X \times X$ is *transitive* if for all $x, y, z \in X$, if $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$.

A relation $R \subseteq X \times X$ is transitive iff $R \circ R \subseteq R$.

A relation $R \subseteq X \times X$ is *reflexive* if $(x, x) \in R$ for all $x \in X$

A relation $R \subseteq X \times X$ is reflexive iff $I_X \subseteq R$.

A relation $R \subseteq X \times X$ is *symmetric* if for all $x, y \in X$, if $(x, y) \in R$, then $(y, x) \in R$

A relation $R \subseteq X \times X$ is symmetric iff $R^{-1} \subseteq R$.

Given $R \subseteq X \times X$ (a relation on X), define R^n by

$$\begin{aligned} R^0 &= I_X \\ R^{n+1} &= R \circ R^n. \end{aligned}$$

The *transtive closure R^+ of R* is given by

$$R^+ = \bigcup_{n \geq 1} R^n.$$

Fact. R^+ is the smallest transitive relation containing R.

The *reflexive and transitive closure R^* of R* is given by

$$R^* = \bigcup_{n \geq 0} R^n = R^+ \cup I_X.$$

Fact. R^* is the smallest transitive and reflexive relation containing R.

A relation $R \subseteq X \times X$ is an *equivalence relation* if it is reflexive, symmetric, and transitive.

Fact. The smallest equivalence relation containing a relation $R \subseteq X \times X$ is given by

$$(R \cup R^{-1})^*.$$

A relation $R \subseteq X \times X$ is *antisymmetric* if for all $x, y \in X$, if $(x, y) \in R$ and $(y, x) \in R$, then $x = y$.

A relation $R \subseteq X \times X$ is a *partial order* if it is reflexive, transitive, and antisymmetic.

A partial order $R \subseteq X \times X$ is a *total order* if for all $x, y \in X$, either $(x, y) \in R$ or $(y, x) \in R$.

2.2 Alphabets, Strings, Languages

Our view of languages is that *a language is a set of strings*.

In turn, a string is a finite sequence of letters from some alphabet. These concepts are defined rigorously as follows.

Definition 2.1. An *alphabet* Σ is any **finite** set.

We often write $\Sigma = \{a_1, \ldots, a_k\}$. The a_i are called the *symbols* of the alphabet.

Examples:

$\Sigma = \{a\}$

$\Sigma = \{a, b, c\}$

$\Sigma = \{0, 1\}$

$\Sigma = \{\alpha, \beta, \gamma, \delta, \epsilon, \lambda, \varphi, \psi, \omega, \mu, \nu, \rho, \sigma, \eta, \xi, \zeta\}$

A string is a finite sequence of symbols. Technically, it is convenient to define strings as functions. For any integer $n \geq 1$, let

$$[n] = \{1, 2, \ldots, n\},$$

and for $n = 0$, let

$$[0] = \emptyset.$$

Definition 2.2. Given an alphabet Σ, a *string over Σ (or simply a string) of length n* is any function

$$u\colon [n] \to \Sigma.$$

The integer n is the *length* of the string u, and it is denoted as $|u|$.

When $n = 0$, the special string $u\colon [0] \to \Sigma$ of length 0 is called the *empty string, or null string*, and is denoted as ϵ.

Given a string $u\colon [n] \to \Sigma$ of length $n \geq 1$, *$u(i)$ is the i-th letter in the string u*. For simplicity of notation, *we write u_i instead of $u(i)$*, and we denote the string $u = u(1)u(2)\cdots u(n)$ as

$$u = u_1 u_2 \cdots u_n,$$

with each $u_i \in \Sigma$.

For example, if $\Sigma = \{a, b\}$ and $u\colon [3] \to \Sigma$ is defined such that $u(1) = a$, $u(2) = b$, and $u(3) = a$, we write

$$u = aba.$$

Other examples of strings are

$$work, \;\; fun, \;\; gabuzomeuh$$

Strings of length 1 are functions $u\colon [1] \to \Sigma$ simply picking some element $u(1) = a_i$ in Σ.

Thus, we will identify every symbol $a_i \in \Sigma$ with the corresponding string of length 1.

The set of all strings over an alphabet Σ, including the empty string, is denoted as Σ^*.

Observe that when $\Sigma = \emptyset$, then

$$\emptyset^* = \{\epsilon\}.$$

When $\Sigma \neq \emptyset$, the set Σ^* is countably infinite. Later on, we will see ways of ordering and enumerating strings.

Strings can be juxtaposed, or concatenated.

Definition 2.3. Given an alphabet Σ, given any two strings $u\colon [m] \to \Sigma$ and $v\colon [n] \to \Sigma$, the *concatenation $u \cdot v$ (also written uv) of u and v* is the string $uv\colon [m+n] \to \Sigma$, defined such that

$$uv(i) = \begin{cases} u(i) & \text{if } 1 \leq i \leq m, \\ v(i-m) & \text{if } m+1 \leq i \leq m+n. \end{cases}$$

In particular, $u\epsilon = \epsilon u = u$. Observe that

$$|uv| = |u| + |v|.$$

For example, if $u = ga$, and $v = buzo$, then

$$uv = gabuzo$$

It is immediately verified that

$$u(vw) = (uv)w.$$

Thus, concatenation is a binary operation on Σ^* which is associative and has ϵ as an identity.

Note that generally, $uv \neq vu$, for example for $u = a$ and $v = b$.

Given a string $u \in \Sigma^*$ and $n \geq 0$, we define u^n recursively as follows:

$$\begin{aligned} u^0 &= \epsilon \\ u^{n+1} &= u^n u \quad (n \geq 0). \end{aligned}$$

Clearly, $u^1 = u$, and it is an easy exercise to show that

$$u^n u = u u^n, \quad \text{for all } n \geq 0.$$

For the induction step, we have

$$\begin{aligned}
u^{n+1} u &= (u^n u) u && \text{by definition of } u^{n+1} \\
&= (u u^n) u && \text{by the induction hypothesis} \\
&= u (u^n u) && \text{by associativity} \\
&= u u^{n+1} && \text{by definition of } u^{n+1}.
\end{aligned}$$

Definition 2.4. Given an alphabet Σ, given any two strings $u, v \in \Sigma^*$ we define the following notions as follows:

u is a prefix of v iff there is some $y \in \Sigma^*$ such that

$$v = uy.$$

u is a suffix of v iff there is some $x \in \Sigma^*$ such that

$$v = xu.$$

u is a substring of v iff there are some $x, y \in \Sigma^*$ such that

$$v = xuy.$$

We say that *u is a proper prefix (suffix, substring) of v* iff u is a prefix (suffix, substring) of v and $u \neq v$.

For example, ga is a prefix of $gabuzo$,

zo is a suffix of $gabuzo$ and

buz is a substring of $gabuzo$.

Recall that a partial ordering $\leq$ on a set S is a binary relation $\leq \subseteq S \times S$ which is reflexive, transitive, and antisymmetric.

The concepts of prefix, suffix, and substring, define binary relations on Σ^* in the obvious way. It can be shown that these relations are partial orderings.

Another important ordering on strings is the lexicographic (or dictionary) ordering.

Definition 2.5. Given an alphabet $\Sigma = \{a_1, \ldots, a_k\}$ assumed totally ordered such that $a_1 < a_2 < \cdots < a_k$, given any two strings $u, v \in \Sigma^*$, we define the *lexicographic ordering* $\preceq$ as follows:

$$u \preceq v \quad \begin{cases} (1) \text{ if } v = uy, \text{ for some } y \in \Sigma^*, \text{ or} \\ (2) \text{ if } u = xa_iy,\ v = xa_jz,\ a_i < a_j, \\ \text{with } a_i, a_j \in \Sigma, \text{ and for some } x, y, z \in \Sigma^*. \end{cases}$$

Note that cases (1) and (2) are mutually exclusive. In case (1) u is a prefix of v. In case (2) $v \not\preceq u$ and $u \neq v$.

For example

$$ab \preceq b, \quad gallhager \preceq gallier.$$

It is fairly tedious to prove that the lexicographic ordering is in fact a partial ordering.

In fact, it is a *total ordering*, which means that for any two strings $u, v \in \Sigma^*$, either $u \preceq v$, or $v \preceq u$.

The *reversal* w^R of a string w is defined inductively as follows:

$$\epsilon^R = \epsilon,$$
$$(ua)^R = au^R,$$

where $a \in \Sigma$ and $u \in \Sigma^*$.

For example
$$reillag = gallier^R.$$

By setting $u = \epsilon$ in $(ua)^R = au^R$ and using the fact that $\epsilon^R = \epsilon$, we obtain $a^R = a$ for all $a \in \Sigma$.

It can be shown that
$$(uv)^R = v^R u^R.$$

Thus,
$$(u_1 \ldots u_n)^R = u_n^R \ldots u_1^R,$$
and when $u_i \in \Sigma$, we have
$$(u_1 \ldots u_n)^R = u_n \ldots u_1.$$

We can now define languages.

Definition 2.6. Given an alphabet Σ, a *language over Σ (or simply a language)* is any subset L of Σ^*.

If $\Sigma \neq \emptyset$, there are uncountably many languages.

A Quick Review of Finite, Infinite, Countable, and Uncountable Sets

For details and proofs, see *Discrete Mathematics*, by Gallier.

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of *natural numbers*.

Recall that a set X is *finite* if there is some natural number $n \in \mathbb{N}$ and a bijection between X and the set $[n] = \{1, 2, \ldots, n\}$. (When $n = 0$, $X = \emptyset$, the empty set.)

The number n is uniquely determined. It is called the *cardinality (or size)* of X and is denoted by $|X|$.

A set is *infinite* iff it is not finite.

Recall that any injection or surjection of a *finite set to itself* is in fact a *bijection*.

The above fails for infinite sets.

The *pigeonhole principle* asserts that *there is no bijection between a finite set X and any proper subset Y of X*.

Consequence: If we think of X as a set of n pigeons and if there are only $m < n$ boxes (corresponding to the elements of Y), then at *least two of the pigeons must share the same box*.

As a consequence of the pigeonhole principle, a set X is infinite iff it is in bijection with a proper subset of itself.

For example, we have a bijection $n \mapsto 2n$ between $\mathbb{N}$ and the set $2\mathbb{N}$ of even natural numbers, a *proper subset* of $\mathbb{N}$, so $\mathbb{N}$ is infinite.

Definition 2.7. A set X is *countable (or denumerable)* if there is an *injection* from X into $\mathbb{N}$.

If X is not the empty set, then X is countable iff there is a *surjection* from $\mathbb{N}$ onto X.

Fact. It can be shown that a set X is countable if either it is finite or if it is in bijection with $\mathbb{N}$ (in which case it is infinite).

We will see later that $\mathbb{N} \times \mathbb{N}$ is countable. As a consequence, the set $\mathbb{Q}$ of rational numbers is countable.

A set is *uncountable* if it is not countable.

For example, $\mathbb{R}$ (the set of real numbers) is uncountable.

Similarly

$$(0, 1) = \{x \in \mathbb{R} \mid 0 < x < 1\}$$

is uncountable. However, there is a bijection between $(0, 1)$ and $\mathbb{R}$ (find one!)

The set $2^{\mathbb{N}}$ of all subsets of $\mathbb{N}$ is uncountable. This is a special case of Cantor's theorem discussed below.

Suppose $|\Sigma| = k$ with $\Sigma = \{a_1, \ldots, a_k\}$.

There are k^n strings of length n and $(k^{n+1} - 1)/(k - 1)$ strings of length at most n over Σ; when $k = 1$, the second formula should be replaced by $n + 1$.

If $\Sigma \neq \emptyset$, then the set Σ^* of all strings over Σ is infinite and countable, as we now show.

If $k = 1$ write $a = a_1$, and then

$$\{a\}^* = \{\epsilon, a, aa, aaa, \ldots, a^n, \ldots\}.$$

We have the bijection $n \mapsto a^n$ from $\mathbb{N}$ to $\{a\}^*$.

If $k \geq 2$, then we can think of the string

$$u = a_{i_1} \cdots a_{i_n}$$

as a representation of the integer $\nu(u)$ in base k shifted by $(k^n - 1)/(k - 1)$, with

$$\begin{aligned} \nu(u) &= i_1 k^{n-1} + i_2 k^{n-2} + \cdots + i_{n-1} k + i_n \\ &= \frac{k^n - 1}{k - 1} + (i_1 - 1)k^{n-1} + \cdots + (i_{n-1} - 1)k + i_n - 1. \end{aligned}$$

(and with $\nu(\epsilon) = 0$), where $1 \leq i_j \leq k$ for $j = 1, \ldots, n$.

We leave it as an exercise to show that $\nu \colon \Sigma^* \to \mathbb{N}$ is a bijection.

In fact, ν corresponds to the enumeration of Σ^* where u precedes v if $|u| < |v|$, and u precedes v in the lexicographic ordering if $|u| = |v|$.

For example, if $k = 2$ and if we write $\Sigma = \{a, b\}$, then the enumeration begins with

$$\begin{array}{llllllll}
\epsilon, \\
0 \\
a, & b, \\
1, & 2, \\
aa, & ab, & ba, & bb, \\
3, & 4, & 5, & 6, \\
aaa, & aab, & aba, & abb, & baa, & bab, & bba, & bbb \\
7, & 8, & 9, & 10, & 11, & 12, & 13, & 14
\end{array}$$

To get the next row, concatenate a on the left, and then concatenate b on the left.

$$\nu(bab) = 2 \cdot 2^2 + 1 \cdot 2^1 + 2 = 8 + 2 + 2 = 12.$$

It works!

On the other hand, if $\Sigma \neq \emptyset$, the set 2^{Σ^*} of all subsets of Σ^* (all languages) is *uncountable*.

Indeed, we can show that *there is no surjection from* $\mathbb{N}$ *onto* 2^{Σ^*}.

First, we will show that *there is no surjection from* Σ^* *onto* 2^{Σ^*}. This is an instance of *Cantor's Theorem*.

We claim that if there is no surjection from Σ^* onto 2^{Σ^*}, then there is no surjection from $\mathbb{N}$ onto 2^{Σ^*} either.

Proof. Assume by contradiction that there is a surjection $g\colon \mathbb{N} \to 2^{\Sigma^*}$.

But, if $\Sigma \neq \emptyset$, then Σ^* is infinite and countable, thus we have the bijection $\nu\colon \Sigma^* \to \mathbb{N}$. Then the composition

$$\Sigma^* \xrightarrow{\nu} \mathbb{N} \xrightarrow{g} 2^{\Sigma^*}$$

is a surjection, because the bijection ν is a surjection, g is a surjection, and the composition of surjections is a surjection, contradicting the hypothesis that there is no surjection from Σ^* onto 2^{Σ^*}. $\square$

We use a *diagonalization* argument to prove *Cantor's Theorem*.

Theorem 2.1. *(Cantor, 1873) For every set X, there is no surjection from X onto 2^X.*

Proof. Assume there is a surjection $h\colon X \to 2^X$, and consider the set

$$D = \{x \in X \mid x \notin h(x)\} \in 2^X.$$

By definition, for any $x \in X$ we have $x \in D$ iff $x \notin h(x)$. Since h is surjective, there is some $y \in X$ such that $h(y) = D$. Then, by definition of D and since $D = h(y)$, we have

$y \in D$ iff $y \notin h(y) = D$,

a contradiction. Therefore, h is not surjective. $\square$

Applying Theorem 2.1 to the case where $X = \Sigma^*$, we deduce that there is no surjection from Σ^* onto 2^{Σ^*}.

Therefore, if $\Sigma \neq \emptyset$, then 2^{Σ^*} is uncountable.

Applying Theorem 2.1 to the case where $X = \mathbb{N}$, we see that there is no surjection from $\mathbb{N}$ onto $2^{\mathbb{N}}$. This shows that $2^{\mathbb{N}}$ is uncountable, as we claimed earlier.

For any set X, by mapping $x \in X$ to $\{x\} \in 2^X$, we obtain an *injection* of X into 2^X. However, Cantor's theorem implies that there is *no injection* of 2^X into X.

Intuitively, 2^X is strictly larger than X.

Since 2^{Σ^*} is uncountable.(if $\Sigma \neq \emptyset$), we will try to single out countable "tractable" families of languages.

We will begin with the family of *regular languages*, and then proceed to the *context-free languages*.

We now turn to operations on languages.

2.3 Operations on Languages

A way of building more complex languages from simpler ones is to combine them using various operations. First, we review the set-theoretic operations of union, intersection, and complementation.

Given some alphabet Σ, for any two languages L_1, L_2 over Σ, the *union* $L_1 \cup L_2$ of L_1 and L_2 is the language

$$L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \quad \text{or} \quad w \in L_2\}.$$

The *intersection* $L_1 \cap L_2$ of L_1 and L_2 is the language

$$L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \quad \text{and} \quad w \in L_2\}.$$

The *difference* $L_1 - L_2$ of L_1 and L_2 is the language

$$L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1 \quad \text{and} \quad w \notin L_2\}.$$

The difference is also called the *relative complement*.

A special case of the difference is obtained when $L_1 = \Sigma^*$, in which case we define the *complement* $\overline{L}$ of a language L as

$$\overline{L} = \{w \in \Sigma^* \mid w \notin L\}.$$

The above operations do not use the structure of strings. The following operations use concatenation.

Definition 2.8. Given an alphabet Σ, for any two languages L_1, L_2 over Σ, the *concatenation L_1L_2 of L_1 and L_2* is the language

$$L_1L_2 = \{w \in \Sigma^* \mid \exists u \in L_1,\ \exists v \in L_2,\ w = uv\}.$$

For any language L, we define L^n as follows:

$$\begin{aligned} L^0 &= \{\epsilon\},\\ L^{n+1} &= L^nL \quad (n \geq 0). \end{aligned}$$

By setting $n = 0$ in the above equation we get $L^1 = L$.

The following properties are easily verified:

$$
\begin{aligned}
L\emptyset &= \emptyset,\\
\emptyset L &= \emptyset,\\
L\{\epsilon\} &= L,\\
\{\epsilon\}L &= L,\\
(L_1 \cup \{\epsilon\})L_2 &= L_1L_2 \cup L_2,\\
L_1(L_2 \cup \{\epsilon\}) &= L_1L_2 \cup L_1,\\
L^nL &= LL^n.
\end{aligned}
$$

In general, $L_1L_2 \neq L_2L_1$.

So far, the operations that we have introduced, except complementation (since $\overline{L} = \Sigma^* - L$ is infinite if L is finite and Σ is nonempty), preserve the finiteness of languages. This is not the case for the next two operations.

Definition 2.9. Given an alphabet Σ, for any language L over Σ, the *Kleene $*$-closure L^* of L* is the language

$$L^* = \bigcup_{n \geq 0} L^n.$$

The *Kleene $+$-closure L^+ of L* is the language

$$L^+ = \bigcup_{n \geq 1} L^n.$$

Thus, L^* is the infinite union

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots,$$

and L^+ is the infinite union

$$L^+ = L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots.$$

Since $L^1 = L$, both L^* and L^+ contain L.

In fact,

$$\begin{aligned} L^+ = \{w \in \Sigma^*,\, &\exists n \geq 1, \\ &\exists u_1 \in L \cdots \exists u_n \in L,\, w = u_1 \cdots u_n\}, \end{aligned}$$

and since $L^0 = \{\epsilon\}$,

$$\begin{aligned} L^* = \{\epsilon\} \cup \{w \in \Sigma^*,\, &\exists n \geq 1, \\ &\exists u_1 \in L \cdots \exists u_n \in L,\, w = u_1 \cdots u_n\}. \end{aligned}$$

Thus, the language L^* always contains ϵ, and we have

$$L^* = L^+ \cup \{\epsilon\}.$$

However, if $\epsilon \notin L$, then $\epsilon \notin L^+$. The following is easily shown:

$$
\begin{aligned}
\emptyset^* &= \{\epsilon\}, \\
L^+ &= L^* L, \\
L^{**} &= L^*, \\
L^* L^* &= L^*.
\end{aligned}
$$

The Kleene closures have many other interesting properties.

Homomorphisms are also very useful.

Given two alphabets Σ, Δ, a *homomorphism* $h\colon \Sigma^* \to \Delta^*$ *between* Σ^* *and* Δ^* is a function $h\colon \Sigma^* \to \Delta^*$ such that

$$
h(uv) = h(u)h(v) \quad \text{for all } u, v \in \Sigma^*.
$$

Letting $u = v = \epsilon$, we get

$$h(\epsilon) = h(\epsilon)h(\epsilon),$$

which implies that (why?)

$$h(\epsilon) = \epsilon.$$

If $\Sigma = \{a_1, \ldots, a_k\}$, it is easily seen that h is completely determined by $h(a_1), \ldots, h(a_k)$ (why?)

Example: $\Sigma = \{a, b, c\}$, $\Delta = \{0, 1\}$, and

$$h(a) = 01, \quad h(b) = 011, \quad h(c) = 0111.$$

For example

$$h(abbc) = 010110110111.$$

Given any language $L_1 \subseteq \Sigma^*$, we define the *image $h(L_1)$ of L_1* as

$$h(L_1) = \{h(u) \in \Delta^* \mid u \in L_1\}.$$

Given any language $L_2 \subseteq \Delta^*$, we define the *inverse image $h^{-1}(L_2)$ of L_2* as

$$h^{-1}(L_2) = \{u \in \Sigma^* \mid h(u) \in L_2\}.$$

We now turn to the first formalism for defining languages, Deterministic Finite Automata (DFA's)