Theory of Computation Class Notes[1]

[1]based on the books by Sudkamp and by Hopcroft, Motwani and Ullman

Contents

List of Figures

Chapter 1

Introduction

1.1 Sets

A *set* is a collection of elements. To indicate that x is an element of the set S, we write $x \in S$. The statement that x is not in S is written as $x \notin S$. A set is specified by enclosing some description of its elements in curly braces; for example, the set of all natural numbers $0, 1, 2, \cdots$ is denoted by

$$\mathcal{N} = \{0, 1, 2, 3, \cdots\}.$$

We use ellipses (i.e.,...) when the meaning is clear, thus $\mathcal{J}_n = \{1, 2, 3, \cdots, n\}$ represents the set of all natural numbers from 1 to n.
When the need arises, we use more explicit notation, in which we write

$$S = \{i | i \geq 0, i \ \textit{is even}\}$$

for the last example. We read this as "S is the set of all i, such that i is greater than zero, and i is even."
Considering a "universal set" U, the complement $\bar{S}$ of S is defined as

$$\bar{S} = \{x | x \in U \wedge x \notin S\}$$

The usual set operations are *union* ($\cup$), *intersection* ($\cap$), and *difference*($-$), defined as

$$S_1 \cup S_2 = \{x | x \in S_1 \vee x \in S_2\}$$
$$S_1 \cap S_2 = \{x | x \in S_1 \wedge x \in S_2\}$$
$$S_1 - S_2 = \{x | x \in S_1 \wedge x \notin S_2\}$$

The set with no elements, called the *empty set* is denoted by $\emptyset$. It is obvious that

$$S \cup \emptyset = S - \emptyset = S$$
$$S \cap \emptyset = \emptyset$$
$$\bar{\emptyset} = U$$
$$\bar{\bar{S}} = S$$

A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S. We write this as

$$S_1 \subseteq S$$

If $S_1 \subseteq S$, but S contains an element not in S_1, we say that S_1 is a *proper subset* of S; we write this as

$$S_1 \subset S$$

The following identities are known as the *de Morgan's laws,*

1. $\overline{S_1 \cup S_2} = \overline{S_1} \cap \overline{S_2}$,
2. $\overline{S_1 \cap S_2} = \overline{S_1} \cup \overline{S_2}$,

1. $\overline{S_1 \cup S_2} = \overline{S_1} \cap \overline{S_2}$,

$$
\begin{array}{ll}
x \in \overline{S_1 \cup S_2} & \\
\quad \Leftrightarrow x \in U \textit{ and } x \notin S_1 \cup S_2 & \\
\quad \Leftrightarrow x \in U \textit{ and } \neg(x \in S_1 \textit{ or } x \in S_2) & (def.union) \\
\quad \Leftrightarrow x \in U \textit{ and } (\neg(x \in S_1) \textit{ and } \neg(x \in S_2)) & (negation\ of\ disjunction) \\
\quad \Leftrightarrow x \in U \textit{ and } (x \notin S_1 \textit{ and } x \notin S_2) & \\
\quad \Leftrightarrow (x \in U \textit{ and } x \notin S_1) \textit{ and } (x \in U \textit{ and } x \notin S_2) & \\
\quad \Leftrightarrow (x \in \overline{S_1} \textit{ and } x \in \overline{S_2}) & (def.complement) \\
\quad \Leftrightarrow x \in \overline{S_1} \cap \overline{S_2} & (def.intersection)
\end{array}
$$

If S_1 and S_2 have no common element, that is,

$$S_1 \cap S_2 = \emptyset,$$

then the sets are said to be *disjoint.*
A set is said to be *finite* if it contains a finite number of elements; otherwise it is *infinite.* The size of a finite set is the number of elements in it; this is denoted by $|S|$ (or $\#S$).
A set may have many subsets. The set of all subsets of a set S is called the *power set* of S and is denoted by 2^S or $P(S)$.
Observe that 2^S is a set of sets.

Example 1.1.1

If S is the set $\{1, 2, 3\}$, then its power set is

$$2^S = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

Here $|S| = 3$ and $|2^S| = 8$. This is an instance of a general result, if S is finite, then

$$|2^S| = 2^{|S|}$$

Proof: (By induction on the number of elements in S).

Basis: $|S| = 1 \Rightarrow 2^S = \{\emptyset, S\} \Rightarrow |2^S| = 2^1 = 2$

Induction Hypothesis: Assume the property holds for all sets S with k elements.

Induction Step: Show that the property holds for (all sets with) $k + 1$ elements.

Denote

$$
\begin{aligned}
S_{k+1} &= \{y_1, y_2, \ldots, y_{k+1}\} \\
&= S_k \cup \{y_{k+1}\}
\end{aligned}
$$

where $S_k = \{y_1, y_2, y_3, \ldots, y_k\}$

$$\begin{aligned} 2^{S_{k+1}} = 2^{S_k} &\cup \{y_{k+1}\} \\ &\cup \{y_1, y_{k+1}\} \cup \{y_2, y_{k+1}\} \cup \ldots \cup \{y_k, y_{k+1}\} \cup \\ &\cup_{x,y \in S_k} \{x, y, y_{k+1}\} \cup \ldots \\ &\cup\ S_{k+1} \end{aligned}$$

2^{S_k} has 2^k elements by the induction hypothesis.

The number of sets in $2^{S_{k+1}}$ which contain y_{k+1} is also 2^k.
Consequently $|2^{S_{k+1}}| = 2 * 2^k = 2^{k+1}$.

A set which has as its elements ordered sequences of elements from other sets is called the *Cartesian product* of the other sets. For the Cartesian product of two sets, which itself is a set of ordered pairs, we write

$$S = S_1 \times S_2 = \{(x, y) \mid x \in S_1,\ y \in S_2\}$$

Example 1.1.2

Let $S_1 = \{1, 2\}$ and $S_2 = \{1, 2, 3\}$. Then

$$S_1 \times S_2 = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3)\}$$

Note that the order in which the elements of a pair are written matters; the pair $(3, 2)$ is not in $S_1 \times S_2$.

Example 1.1.3

If A is the set of throws of a coin, i.e., $A =$\{head,tail\}, then

$$A \times A = \{\text{(head,head),(head,tail),(tail,head),(tail,tail)}\}$$

the set of all possible throws of two coins.
The notation is extended in an obvious fashion to the Cartesian product of more than two sets; generally

$$S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \cdots, x_n) \mid x_i \in S_i\}$$

1.2 Functions and Relations

A *function* is a rule that assigns to elements of one set (the function *domain*) a unique element of another set (the *range*). We write

$$f : S_1 \to S_2$$

to indicate that the domain of the function f is a subset of S_1 and that the range of f is a subset of S_2. If the domain of f is all of S_1, we say that f is a *total function* on S_1; otherwise f is said to be a *partial function* on S_1.

1. *Domain* $f = \{x \in S_1 \mid (x, y) \in f$, for some $y \in S_2\} = D_f$
2. *Range* $f = \{y \in S_2 \mid (x, y) \in f$, for some $x \in S_1\} = R_f$

3. The *restriction* of f *to* $A \subseteq S_1$, $f_{|A} = \{(x, y) \in f \mid x \in A\}$
4. The *inverse* $f^{-1} : S_2 \to S_1$ *is* $\{(y, x) \mid (x, y) \in f\}$
5. $f : S_1 \to S_1$ is called a *function on* S_1
6. If $x \in D_f$ then f is *defined* at x; otherwise f is *undefined* at x;
7. f is a *total* function if $D_f = S_1$.
8. f is a *partial* function if $D_f \subseteq S_1$
9. f is an *onto* function or *surjection* if $R_f = S_2$. If $R_f \subseteq S_2$ then f is a function from $S_1(D_f)$ *into* S_2
10. f is a *one to one* function or *injection* if $(f(x) = z$ and $f(y) = z) \Rightarrow x = y$
11. A total function f is a *bijection* if it is both an injection and a surjection.

    A function can be represented by a set of pairs $\{(x_1, y_1), (x_2, y_2), \cdots\}$, where each x_i is an element in the domain of the function, and y_i is the corresponding value in its range. For such a set to define a function, each x_i can occur at most once as the first element of a pair. If this is not satisfied, such a set is called a *relation*.

A specific kind of relation is an *equivalence relation*. A relation denoted r on X is an equivalence relation if it satisfies three rules,
the *reflexivity* rule:

$$(x, x) \in r \quad \forall x \in \mathrm{X}$$

the *symmetry* rule:

$$(x, y) \in r \text{ then } (y, x) \in r \quad \forall x, y \in \mathrm{X}$$

and
the *transitivity* rule:

$$(x, y) \in r, (y, z) \in r \text{ then } (x, z) \in r \quad \forall x, y, z \in \mathrm{X}$$

An equivalence relation on X induces a *partition* on X into disjoint subsets called *equivalence classes* X_j, $\cup_j X_j = X$, such that elements from the same class belong to the relation, and any two elements taken from different classes are not in the relation.

Example 1.2.1

The relation congruence mod m (modulo m) on the set of the integers $\mathcal{Z}$.
$i = j \bmod m$ if $i - j$ is divisible by m; $\mathcal{Z}$ is partitioned into m equivalence classes:

$$\begin{gathered}\{\cdots, -2m, -m, 0, m, 2m, \cdots\} \\ \{\cdots, -2m+1, -m+1, 1, m+1, 2m+1, \cdots\} \\ \{\cdots, -2m+2, -m+2, 2, m+2, 2m+2, \cdots\} \\ \cdots \\ \{\cdots, -m-1, -1, m-1, 2m, 3m-1, \cdots\}\end{gathered}$$

1.3 Countable and uncountable sets

Cardinality is a measure that compares the size of sets. The cardinality of a finite set is the number of elements in it. The cardinality of a finite set can thus be obtained by counting the elements of the set.Two sets X and Y have the same cardinality if there is a total one to one function from X onto Y (i.e., a *bijection* from X to Y). The cardinality of a set X is less than or equal to the cardinality of a set Y if there is a total one to one function from X into Y. We denote cardinality of X by $\#X$ or $|X|$.
A set that has the same cardinality as the set of natural numbers $\mathcal{N}$, is said to be *countably infinite* or *denumerable*. Sets that are either finite or denumerable are referred to as *countable* sets. The elements of a countably infinite set can be indexed (or enumerated) using $\mathcal{N}$ as the index set. The index mapping yields an *enumeration* of the countably infinite set. Sets that are not countable are said to be *uncountable*.

- The cardinality of denumerable sets is $\#\mathcal{N} = \aleph_0$ ("$aleph_0$")
- The cardinality of the set of the real numbers, $\#\mathcal{R} = \aleph_1$ ("$aleph_1$")

A set is infinite if it has proper subset of the same cardinality.

Example 1.3.1

The set $\mathcal{J} = \mathcal{N} - \{0\}$ is countably infinite; the function $s(n) = n + 1$ defines a one-to-one mapping from $\mathcal{N}$ onto $\mathcal{J}$. The set $\mathcal{J}$, obtained by removing an element from $\mathcal{N}$, has the same cardinality as $\mathcal{N}$. Clearly, there is no one to one mapping of a finite set onto a proper subset of itself. It is this property that differentiates finite and infinite sets.

Example 1.3.2

The set of odd natural numbers is *denumerable*. The function $f(n) = 2n + 1$ establishes the bijection between $\mathcal{N}$ and the set of the odd natural numbers.

The one to one correspondence between the natural numbers and the set of all integers exhibits the countability of set of integers. A correspondence is defined by the function

$$f(n) = \begin{cases} \lfloor \frac{n}{2} \rfloor + 1 & \text{if } n \text{ is odd} \\ -\frac{n}{2} & \text{if } n \text{ is even} \end{cases}$$

Example 1.3.3

$\#\mathcal{Q}^+ = \#\mathcal{J} = \#\mathcal{N}$
$\mathcal{Q}^+$ is the set of the rational numbers $\frac{p}{q} > 0$, where p and q are integers, $q \neq 0$.

1.4 Proof Techniques

We will give examples of proof by induction, proof by contradiction, and proof by Cantor diagonalization.
In proof by induction, we have a sequence of statements $P_1, P_2, \cdots$, about which we want to make some claim. Suppose that we know that the claim holds for all statements $P_1, P_2, \cdots$, up to P_n. We then try to argue that this implies that the claim also holds for P_{n+1}. If we can carry out this inductive step for all positive n, and if we have some starting point for the induction, we can say that the claim holds for all statements in the sequence.
The starting point for an induction is called the *basis*. The assumption that the claim holds for

statements $P_1, P_2, \cdots, P_n$ is the *induction hypothesis*, and the argument connecting the induction hypothesis to P_{n+1} is the *induction step*. Inductive arguments become clearer if we explicitly show these three parts.

Example 1.4.1 *Let us prove*

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

by mathematical induction. We establish

(a) the basis by substituting 0 for n in $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ and observing that both sides are 0.

(b) For the induction hypothesis, we assume that the property holds with $n = k$;

$$\sum_{i=0}^{k} i^2 = \frac{k(k+1)(2k+1)}{6}$$

(c) In the induction step, we show that the property holds for $n = k + 1$; i.e.,

$$\sum_{i=0}^{k} i^2 = \frac{(k)(k+1)(2k+1)}{6}$$

$$\Rightarrow \sum_{i=0}^{k+1} i^2 = \frac{(k+1)(k+2)(2k+3)}{6}$$

Since

$$\sum_{i=0}^{k+1} i^2 = \sum_{i=0}^{k} i^2 + (k+1)^2$$

and in view of the induction hypothesis, we need only show that

$$\frac{(k)(k+1)(2k+1)}{6} + (k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6}$$

The latter equality follows from simple algebraic manipulation.

In a proof by contradiction, we assume the opposite or contrary of the property to be proved; then we prove that the assumption is invalid.

Example 1.4.2

Show that $\sqrt{2}$ is not a rational number.
As in all proofs by contradiction, we assume the contrary of what we want to show. Here we assume that $\sqrt{2}$ is a rational number so that it can be written as

$$\sqrt{2} = \frac{n}{m},$$

where n and m are integers without a common factor. Rearranging ($\sqrt{2} = \frac{n}{m}$), we have

$$2m^2 = n^2$$

Therefore n^2 must be even. This implies that n is even, so that we can write $n = 2k$ or

$$2m^2 = 4k^2$$

and

$$m^2 = 2k^2$$

Therefore m is even. But this contradicts our assumption that n and m have no common factor. Thus, m and n in $(\sqrt{2} = \frac{n}{m})$ cannot exist and $\sqrt{2}$ is not a rational number.

This example exhibits the essence of a proof by contradiction. By making a certain assumption we are led to a contradiction of the assumption or some known fact. If all steps in our argument are logically sound, we must conclude that our initial assumption was false.

To illustrate **Cantor's diagonalization** method, we prove that the set $A = \{f | f$ a total function, $f : \mathcal{N} \to \mathcal{N}\}$, is uncountable. This is essentially a proof by contradiction; so we assume that A is countable, i.e., we can give an enumeration $f_0, f_1, f_2, \cdots$ of A. To come to a contradiction, we construct a new function $\overline{f}$ as

$$\overline{f}(x) = f_x(x) + 1 \qquad x \in \mathcal{N}$$

The function $\overline{f}$ is constructed from the diagonal of the function values of $f_i \in A$ as represented in the figure below. For each $x, \overline{f}$ differs from f_x on input x. Hence $\overline{f}$ does not appear in the given enumeration. However $\overline{f}$ is total and $\overline{f} : \mathcal{N} \to \mathcal{N}$. Such an $\overline{f}$ can be given for any chosen enumeration. This leads to a contradiction.
Therefore A cannot be enumerated; hence A is uncountable.

$$\begin{array}{cccc} f_0 & f_0(0) & f_0(1) & f_0(2)\cdots \\ f_1 & f_1(0) & f_1(1) & f_1(2)\cdots \\ f_2 & f_2(0) & f_2(1) & f_2(2)\cdots \\ f_3 & f_3(0) & f_3(1) & f_3(2)\cdots \end{array}$$

Remarks:
The set of all infinite sequences of 0's and 1's is uncountable. With each infinite sequence of 0's and 1's we can associate a real number in the range $[0, 1)$. As a consequence, the set of real numbers in the range $[0, 1)$ is uncountable. Note that the set of all real numbers is also uncountable.

Chapter 2

Languages and Grammars

2.1 Languages

We start with a finite, nonempty set Σ of symbols, called the alphabet. From the individual symbols we construct *strings* (over Σ or on Σ), which are finite sequences of symbols from the alphabet. The empty string ε is a string with no symbols at all. Any set of strings over/on Σ is a *language* over/on Σ.

Example 2.1.1

$$\begin{aligned}\Sigma &= \{c\}\\ L_1 &= \{cc\}\\ L_2 &= \{c, cc, ccc\}\\ L_3 &= \{w | w = c^k, k = 0, 1, 2, \ldots\}\\ &= \{\varepsilon, c, cc, ccc, \ldots\}\end{aligned}$$

Example 2.1.2

$$\begin{aligned}\Sigma &= \{a, b\}\\ L_1 &= \{ab, ba, aa, bb, \varepsilon\}\\ L_2 &= \{w | w = (ab)^k, k = 0, 1, 2, 3, \ldots\}\\ &= \{\varepsilon, ab, abab, ababab, \ldots\}\end{aligned}$$

The *concatenation* of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \ldots a_n$$

and

$$v = b_1 b_2 \ldots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_m$$

If w is a string, then w^n is the string obtained by concatening w with itself n times. As a special case, we define

$$w^0 = \varepsilon,$$

for all w. Note that $\varepsilon w = w\varepsilon = w$ for all w. The *reverse* of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \dots a_2 a_1$$

If

$$w = uv,$$

then u is said to be *prefix* and v a *suffix* of w.
The *length* of a string w, denoted by $|w|$, is the number of symbols in the string.
Note that,

$$|\varepsilon| = 0$$

If u and v are strings, then the length of their concatenation is the sum of the individual lengths,

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

Let us show that $|uv| = |u| + |v|$. To prove this by induction on the length of strings, let us define the length of a string recursively, by

$$|a| = 1$$

$$|wa| = |w| + 1$$

for all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive understanding of the length of a string: the length of a single symbol is one, and the length of any string is incremented by one if we add another symbol to it.

Basis: $|uv| = |u| + |v|$ holds for all u of any length and all v of length 1 (by definition).

Induction Hypothesis: we assume that $|uv| = |u| + |v|$ holds for all u of any length and all v of length $1, 2, \dots, n$.

Induction Step: Take any v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$

$$|uv| = |uwa| = |uw| + 1.$$

By the induction hypothesis (which is applicable since w is of length n).

$$|uw| = |u| + |w|.$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v|.$$

which completes the induction step.

If Σ is an alphabet, then we use Σ^* to denote the set of strings obtained by concatenating zero or more symbols from Σ. We denote $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. The sets Σ^* and Σ^+ are always infinite.
A *language* can thus be defined as a subset of Σ^*. A string w in a language L is also called a *word* or a *sentence* of L.

Example 2.1.3

$\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

The set

$$\{a, aa, aab\}.$$

is a language on Σ. Because it has a finite number of words, we call it a finite language. The set

$$L = \{a^n b^n | n \geq 0\}$$

is also a language on Σ. The strings *aabb* and *aaaabbbb* are words in the language L, but the string *abb* is not in L. This language is infinite.
Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to Σ^*; that is, the complement of L is

$$\overline{L} = \Sigma^* - L$$

The concatenation of two languages L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

We define L^n as L concatenated with itself n times, with the special case

$$L^0 = \{\varepsilon\}$$

for every language L.

Example 2.1.4

$L_1 = \{a, aaa\}$
$L_2 = \{b, bbb\}$

$$L_1 L_2 = \{ab, abbb, aaab, aaabbb\}$$

Example 2.1.5

For

$$L = \{a^n b^n | n \geq 0\},$$

then

$$L \cdot L = L^2 = \{a^n b^n a^m b^m | n \geq 0, m \geq 0\}$$

The string *aabbaaabbb* is in L^2.The *star-closure* or *Kleene closure* of a language is defined as

$$\begin{aligned} L^* &= L^0 \cup L^1 \cup L^2 \cdots \\ &= \bigcup_{i=0}^{\infty} L^i \end{aligned}$$

and the *positive closure* as

$$\begin{aligned} L^+ &= L^1 \cup L^2 \cdots \\ &= \bigcup_{i=1}^{\infty} L^i \end{aligned}$$

2.2 Regular Expressions

Definition 2.2.1 *Let Σ be a given alphabet. Then,*

1. *$\emptyset$, $\{\varepsilon\}$, and $\{a\}$ $\forall a \in \Sigma$ are regular sets. They are called primitive regular sets.*
2. *If S and S_1 are regular sets, so are S^*, $X \cup Y$ and $X \centerdot Y$.*
3. *A set is a regular set if it is a primitive regular set or can be derived from the primitive regular sets by applying a finite number of the operations cup, * and* concatenation.

Definition 2.2.2 *Let Σ be a given alphabet. Then,*

1. *$\emptyset$, ε (representing $\{\varepsilon\}$), a (representing $\{a\}$) $\forall a \in \Sigma$ are regular expressions. They are called primitive regular expressions.*
2. *If r and r_1 are regular expressions so are (r), (r^*), $(r_1 + r_2)$, $(r \centerdot r_1)$.*
3. *A string is a regular expression if it is a primitive regular expression or can be derived from the primitive regular expressions by applying a finite number of the operations +, * and* concatenation.

A regular expression denotes a regular set.
Regarding the notation of regular expression, texts will usually print them boldface; however, we assume that it will be understood that, in the context of regular expressions, ε is used to represent $\{\varepsilon\}$ and a is used to represent {a}.

Example 2.2.1

$b^*(ab^*ab^*)$ is a regular expression.

Example 2.2.2

$$\begin{aligned}(c + da^*bb)^* &= \{c, dbb, dabb, daabb, \ldots\}^* \\ &= \{\varepsilon, c, cc, \ldots, dbb, dbbdbb, \ldots, dabb, dabbdabb, \ldots, cdbb, cdabb, \ldots\}\end{aligned}$$

Beyond the usual properties of + and concatenation, important equivalences involving *regular expressions* concern porperties of the closure (Kleene star) operation. Some are given below, where α, β, γ stand for arbitrary regular expressions:

1. $(\alpha^*)^* = \alpha^*$.
2. $(\alpha\alpha^*) = \alpha^*\alpha$.
3. $\alpha\alpha^* + \varepsilon = \alpha^*$.
4. $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$.
5. $\alpha(\beta\alpha)^* = (\alpha\beta)^*\alpha$.
6. $(\alpha + \beta)^* = (\alpha^* + \beta^*)^*$.
7. $(\alpha + \beta)^* = (\alpha^*\beta^*)^*$.
8. $(\alpha + \beta)^* = \alpha^*(\beta\alpha^*)^*$.

In general, the distribution law does not hold for the closure operation. For example, the statement $\alpha^* + \beta^* \stackrel{?}{=} (\alpha + \beta)^*$ is false because the right hand side denotes no string in which both α and β appear.

2.3 Grammars

Definition 2.3.1 *A* grammar G *is defined as a quadruple*

$$G = (V, \Sigma, S, P)$$

where
V *is a finite set of symbols called* variables *or* nonterminals*,*
Σ *is a finite set of symbols called* terminal symbols *or* terminals*,*
$S \in V$ *is a special symbol called the* start symbol*,*
P *is a finite set of* productions *or* rules *or* production rules*.*

We assume V and Σ are non-empty and disjoint sets.
Production rules specify the transformation of one string into another. They are of the form

$$x \rightarrow y$$

where

$$x \in (V \cup \Sigma)^+ - \Sigma \text{ and}$$

$$y \in (V \cup \Sigma)^*.$$

Given a string w of the form

$$w = uxv$$

we say that the production $x \rightarrow y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string,

$$w \Rightarrow z;$$

we say that w *derives* z or that z is *derived* from w.

Successive strings are derived by applying the productions of the grammar in arbitrary order. A production can be used whenever it is applicable, and it can be applied as often as desired. If

$$w_1 \Rightarrow w_2 \Rightarrow w_3 \cdots \Rightarrow w$$

we say that w_1 derives w, and write $w_1 \overset{*}{\Rightarrow} w$.

The * indicates that an unspecified number of steps (including zero) can be taken to derive w from w_1. Thus

$$w \Rightarrow w$$

is always the case. If we want to indicate that atleast one production must be applied, we can write

$$w \overset{+}{\Rightarrow} v$$

Let $G = (V, \Sigma, S, P)$ be a grammar. Then the set

$$L(G) = \{w \in \Sigma^* | s \overset{*}{\Rightarrow} w\}$$

is the language generated by G. If $w \in L(G)$, then the sequence

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w$$

is a *derivation* of the sentence (or word) w. The strings $S, w_1, w_2, \cdots$, are called *sentential forms* of the derivation.

Example 2.3.1

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P)$$

with P given by,

$$S \to aSb$$

$$S \to \varepsilon$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

The string $aabb$ is a sentence in the language generated by G.

Example 2.3.2

P:

$$< sentence > \to < Noun\_phrase >< Verb\_phrase >$$
$$< Noun\_phrase > \to < Determiner >< Noun\_phrase > \mid < Adjective >< Noun >$$
$$< Noun\_phrase > \to < Article >< Noun >$$
$$< Verb\_phrase > \to < Verb >< Noun\_phrase >$$
$$< Determiner > \to This$$
$$< Adjective > \to Old$$
$$< Noun > \to Man|Bus$$
$$< Verb > \to Missed$$
$$< Article > \to The$$

Example 2.3.3

$< expression > \Rightarrow < variable > \mid < expression >< operation >< expression >$
$< variable > \to A|B|C| \cdots |Z$
$< operation > \to +| - | * |/$

Leftmost Derivation

$$\begin{aligned} < expression > &\to < expression >< operation >< expression > \\ &\Rightarrow < variable >< operation >< expression > \\ &\Rightarrow A < operation >< expression > \\ &\Rightarrow A+ < expression > \\ &\Rightarrow A+ < expression >< operation >< expression > \\ &\Rightarrow A+ < variable >< operation >< variable > \\ &\Rightarrow A + B < operation >< expression > \\ &\Rightarrow A + B * < expression > \\ &\Rightarrow A + B * < variable > \\ &\Rightarrow A + B * C \end{aligned}$$

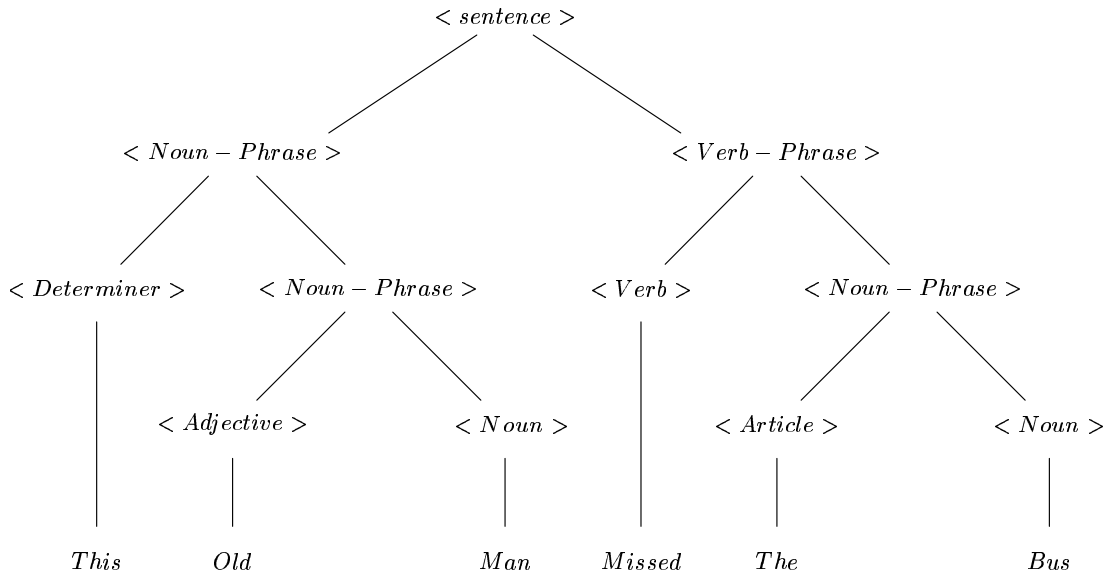

Figure 2.1: Derivation tree

This is a leftmost derivation of the string $A + B * C$ in the grammar (corresponding to $A + (B * C)$). Note that another leftmost derivation can be given for the above expression.
A grammar G (such as the one above) is called *ambiguous* if some string in $L(G)$ has more than one leftmost derivation. An unambiguous grammar for the language is the following:

$$< expr > \rightarrow < multi - expr > \mid < multi - expr >< add - expr >< expr >$$
$$< multi - expr > \rightarrow < variable > \mid < variable >< multi - op >< variable >$$
$$< multi - op > \rightarrow * \mid /$$
$$< add - op > \rightarrow + \mid -$$
$$< variable > \rightarrow A \mid B \mid C \mid \cdots \mid Z$$

Note that, for an inherently ambiguous language L, every grammar that generates L is ambiguous.

Example 2.3.4

$$G : S \rightarrow \varepsilon \mid aSb \mid bSa \mid SS$$

$$L = \{w | n_a(w) = n_b(w)\}$$

Show that $L(G) = L$

1. $L(G) \subseteq L$. (All strings derived by G, are in L.)
   For $w \in L(G)$, all productions of G add a number of a's which is same as the number of b's added;

   $$\Rightarrow n_a(w) = n_b(w)$$

   $$\Rightarrow w \in L$$

2. $L \subseteq L(G)$
   Let $w \in L$. By definition of L, $n_a(w) = n_b(w)$. We show that $w \in L(G)$ by induction (on the

length of w).

Basis: ε is in both L and $L(G)$.
$|w| = 2$. The only two strings of length 2 in L are ab and ba
$S \Rightarrow aSb$
$\Rightarrow ab$
$S \Rightarrow bSa$
$\Rightarrow ba$

Induction Hypothesis: $\forall w \in L$ with $2 \leq |w| \leq 2i$, we assume that $w \in L(G)$.

Induction Step: Let $w_1 \in L$, $|w_1| = 2i + 2$.

(a) w_1 of the form
$w_1 = awb$ (or bwa) where $|w| = 2i$
$\Rightarrow w \in L(G)$ (by I. H.)
We derive $w_1 = awb$ using the rule $S \rightarrow aSb$.
We derive $w_1 = bwa$ using the rule $S \rightarrow bSa$.

(b) $w_1 = awa$ or
$w_1 = bwb$
Let us assign a count of +1 to a and -1 to b;
Thus for $w_1 \in L$ the total count = 0.

We will now show that count goes through 0 at least once within $w_1 = awa$ (case bwb is similar)
$w_1 = a$ (count = +1) (count goes through 0) (count = -1) a (by end, count = 0).
$\Rightarrow w_1 = w'$ $(count = 0)$ w'' where
$w' \in L$,
$w'' \in L$.

We also have $|w''| \geq 2$ and $|w'| \geq 2$ so that

$|w'| \leq 2i$ and
$|w''| \leq 2i$
$\Rightarrow w', w'' \in L(G)$ (I. H.)
$w_1 = w'w''$ can be derived in G from w' and w'', using the rule $S \rightarrow SS$.

Example 2.3.5

$L(G) = \{a^{2^n} | n \geq 0\}$
$G = (V, T, S, P)$ *where*
$V = \{S, [,], A, D\}$
$T = \{a\}$

P :$S \rightarrow [A]$
$[\rightarrow [D \mid \varepsilon$
$D] \rightarrow]$
$DA \rightarrow AAD$
$] \rightarrow \varepsilon$
$A \rightarrow a$

For example, let us derive a^4.

$S \Rightarrow [A]$
$\Rightarrow [DA]$
$\Rightarrow [AAD]$
$\Rightarrow [AA]$
$\Rightarrow [DAA]$
$\Rightarrow [AADA]$
$\Rightarrow [AAAAD]$
$\Rightarrow [AAAA]$
$\Rightarrow \varepsilon AAAA\varepsilon$
$\Rightarrow AAAA$
$\Rightarrow aaaa$
$\Rightarrow a^4$

Example 2.3.6

$L(G) = \{w \in \{a,b,c\}^* \mid n_a(w) = n_b(w) = n_c(w)\}$
$V = \{A, B, C, S\}$
$T = \{a, b, c\}$
$P : S \rightarrow \varepsilon | ABCS$
$AB \rightarrow BA$
$AC \rightarrow CA$
$BC \rightarrow CB$
$BA \rightarrow AB$
$CA \rightarrow AC$
$CB \rightarrow BC$
$A \rightarrow a$
$B \rightarrow b$
$C \rightarrow c$
derive *ccbaba*

Solution:

$S \Rightarrow ABCS$
$\Rightarrow ABCABCS$
$\Rightarrow ABCABC\varepsilon$
$\Rightarrow A\underline{BC}A\underline{BC}$
$\Rightarrow \underline{AC}B\underline{AC}B$
$\Rightarrow CA\underline{BCAB}$
$\Rightarrow C\underline{AC}BBA$
$\Rightarrow CC\underline{AB}BA$
$\Rightarrow CCBABA$
$\Rightarrow ccbaba$

Example 2.3.7

$S \rightarrow \varepsilon \mid aSb$
$L(G) = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$
$L = \{a^i b^i | i \geq 0\}$

To prove that $L = L(G)$
1. $L(G) \subseteq L$
2. $L \subseteq L(G)$

2. $L \subseteq L(G)$:
   Let $w \in L$, $w = a^k b^k$
   we apply $S \to aSb$ (k times), thus

$$S \Rightarrow a^k S b^k$$

   then $S \to \varepsilon$

$$S \Rightarrow a^k b^k$$

1. $L(G) \subseteq L$:
   We need to show that, if w can be derived in G, then $w \in L$. ε is in the language, by definition. We first show that all sentential forms are of the form $a^i S b^i$, by induction on the length of the sentential form.

   Basis: $(i = 1)$ aSb is a sentential form, since $S \Rightarrow aSb$.

   Induction Hypothesis: Sentential form of length $2i + 1$ is of the form $a^i S b^i$.

   Induction Step: Sentential form of length $2(i + 1) + 1 = 2i + 3$ is derived as

$$S \Rightarrow aSb \Rightarrow a(a^i S b^i)b = a^{i+1} S b^{i+1}.$$

   To get a sentence, we must apply the production $S \to \varepsilon$; i.e.,

$$S \Rightarrow a^i S b^i \Rightarrow a^i b^i$$

   represents all possible derivations; hence G derives only strings of the form $a^i b^i$ $(i \geq 0)$.

2.4 Classification of Grammars and Languages

A classification of grammars (and the corresponding classes of languages) is given with respect to the form of the grammar rules $x \to y$, into the *Type 0, Type 1, Type 2* and *Type 3* classes.

Type 0 *Unrestricted* grammars do not put restrictions on the production rules.

Type 1 If all the grammar rules $x \to y$ satisfy $|x| \leq |y|$, then the grammar is *context sensitive* or Type 1. Grammar G will generate a language L(G) which is called a *context-sensitive* language. Note that x has to be of length at least 1 and thereby y too. Hence, it is not possible to derive the empty string in such a grammar.

Type 2 If all production rules are of the form $x \to y$ where $|x| = 1$, then the grammar is said to be *context-free* or Type 2 (i.e., the left hand side of each rule is of length 1).

Type 3 If the production rules are of the following forms:
$A \to xB$
$A \to x$
where $x \in \Sigma^*$ (a string of all terminals or the empty string),
and $A, B \in V$ (variables),
then the grammar is called *right linear*.

Similarly, for a *left linear* grammar, the production rules are of the form
$A \to Bx$
$A \to x$

For a *regular grammar*, the production rules are of the form
$A \to aB$
$A \to a$
$A \to \varepsilon$
with $a \in \Sigma$.

A language which can be generated by a regular grammar will (later) be shown to be regular. Note that, a language that can be derived by a regular grammar iff it can be derived by a right linear grammar iff it can be derived by a left linear grammar.

2.5 Normal Forms of Context-Free Grammars

2.5.1 Chomsky Normal Form (CNF)

Definition 2.5.1 *A context-free grammar $G = (V, \Sigma, P, S)$ is in **Chomsky Normal Form** if each rule is of the form*

i) $A \to BC$

ii) $A \to a$

iii) $S \to \varepsilon$

where $B, C \in V - \{S\}$

Theorem 2.5.1 *Let $G = (V, \Sigma, P, S)$ be a context-free grammar. There is an algorithm to construct a grammar $G' = (V, \Sigma, P', S)$ in Chomsky normal form that is equivalent to G ($L(G') = L(G)$).*

Example 2.5.1

Convert the given grammar G to CNF.

$$\begin{aligned} G : S &\to aABC|a \\ A &\to aA|a \\ B &\to bcB|bc \\ C &\to cC|c \end{aligned}$$

Solution:

A CNF equivalent G' can be given as :

$G' : S \to A'T_1|a$
$A' \to a$
$T_1 \to AT_2$
$T_2 \to BC$
$A \to A'A$
$B \to B'T_3|B'C'$
$B' \to b$
$T_3 \to C'B$
$C \to C'C|c$
$C' \to c$

2.5.2 Greibach Normal Form (GNF)

If a grammar is in GNF, then the length of the terminals prefix of the sentential form is increased at every grammar rule application, thereby enabling the prevention of the left recursion.

Definition 2.5.2 *A context-free grammar* $G = (V, \Sigma, P, S)$ *is in* ***Greibach Normal Form*** *if each rule is of the form,*

i) $A \to aA_1A_2 \dots A_n$

ii) $A \to a$

iii) $S \to \varepsilon$

Chapter 3

Finite State Automata

Bibliography

[1] A. Thomas Sudkamp, *An Introduction to the Theory of Computer Science, Languages and Machnes*, 3rd edition, Addison-Wesley 2005.

[2] John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 2nd edition, Addison-Wesley 2001.